VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Server pro podporu výuky ARUO

Supporting Server for Constraint Processing Teaching

2017 Bc. Jana Legerská

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jana Legerská**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Server pro podporu výuky ARUO
Supporting Server for Constraint Processing Teaching

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je vytvořit server, na kterém si studenti budou moci spustit výpočet vybraných algoritmů z oblasti constraint processing na jimi zadaných i ukázkových vstupech a na zobrazeném průběhu výpočtu lépe pochopit princip fungování těchto algoritmů. Tato práce se konkrétně zaměří na varianty metody eliminace košíků (bucket elimination).

1. Nastudujte si algoritmus eliminace košíků (bucket elimination) pro optimalizační úlohy a jeho variantu eliminace minikošíků, použití pro počítání počtu řešení, použití pro úlohy na pravděpodobnostních sítích a tyto algoritmy naprogramujte .
2. Cílem není nejefektivnější implementace algoritmů, ale taková, která umožní zobrazovat postupně jednotlivé kroky tak, aby uživatel mohl pochopit princip jejich fungování.
3. Vytvořte výukový server, kde bude možné zadat vstupní data pro naprogamované algoritmy a pro tato data si nechat zobrazit postup výpočtu.
4. Vytvořte i ukázkové vstupy pro jednotlivé algoritmy, aby postup výpočtu bylo možné zobrazit i bez zadávání vstupu uživatelem.

Seznam doporučené odborné literatury:

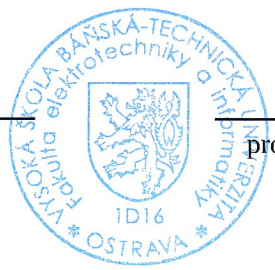
Rina Dechter: Constraint Processing, Morgan Kaufmann Publishers 2003, ISBN: 978-1-55860-890-0

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 14.07.2017



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 13.července 2017 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 13.července 2017 .....

V souvislosti s vypracováním této práce bych ráda poděkovala panu Ing. Martinu Kotovi, Ph.D. za odbornou pomoc, a také za připomínky a návrhy, které pomáhaly dotvářet tuto práci.

Abstrakt

Tato diplomová práce je zaměřena na tvorbu softwaru pro účel podpory výuky předmětu ARUO. Zabývá se řešením z oblasti "Constraint processing" a jeho metody Eliminace košíku. Práce především řeší eliminaci košíku, a jeho varianty eliminace minikošíků, eliminace košíku pro optimalizaci, použití pro počítání počtu řešení a použití pro úlohy pravděpodobnostních síti (metodu MPE). Aplikace běží na webovém serveru, a uživatelé si řešení zobrazují ve webovém prohlížeči. Aplikace není implementována nejefektivněji, ale slouží na zobrazení celého postupu na uživatelem zadaném příkladu.

Klíčová slova: eliminace košíku, řešení úloh s omezeními, optimalizační úlohy, pravděpodobnostní sítě, síť omezení, ARUO

Abstract

This thesis is focused on creating software for the purposes of subject ARUO. This thesis deals with solving Constraint processing and its methods Bucket elimination. This thesis solves Bucket elimination for optimization, counting the member of solutions, mini-bucket elimination and Bucket elimination for MPE. The application is running on web server and users use Web browser to view solutions. Application is not implemented effectively, but application used to view of the gradual calculation.

Key Words: bucket elimination, constraint processing, constraint optimization, probablistic networks, belief network, constraint network, ARUO

Obsah

Seznam použitých zkratek a symbolů

SW	–	Software
ARUO	–	Automatizované řešení úloh s omezeními
CP	–	Constraint processing
CN	–	Constrain network
KNF	–	Konjuktivní normální forma
SAT	–	Satisfiability
COP	–	Constraint optimization problem
CSP	–	Constraint satistaction problem
MPE	–	The most probable explanation
MPA	–	The maximum a posteriori hypothesis
JSP	–	JavaServlet Pages
CSS	–	Cascading Style Sheets
PDF	–	Portable Document Format

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

ARUO nebo také Automatizované řešení úloh s omezeními je předmět vyučovaný na VŠB - technické univerzitě Ostrava. Cílem tohoto předmětu je představit studentům na zvolených praktických problémech možné způsoby popisu omezení a vybrané metody hledání řešení vyhovujících těmto omezením. Práce je zaměřena na oblast Constraint processingu (úlohy s omezeními). Tyto úlohy používáme v běžném životě, aniž bychom si to uvědomovali. Kdykoliv chceme něco plánovat, či řešit nějaký problém, můžeme k tomuto řešení použít constraint procesing a jeho metody řešení.

Úkolem je vytvořit výukový server pro předmět ARUO, konkrétně zaměřen na Constraint Processing a jeho metodu Eliminaci košíku. Není úkolem naprogramovat efektivní algoritmus, ale algoritmus, který bude uživateli ukazovat každý krok výpočtu. Uživatel pak bude schopen na svých zadáních vidět jak by měl výpočet probíhat krok po kroku, a pochopit jeho funkčnost.

Cílem této práce bude vytvoření webové aplikace programované v jazyce Java s již zadanými ukázkovými příklady a s možností vložení vlastního vstupu. Uživatelé budou schopni se blíže podívat na metody Eliminace košíku pro optimalizaci, využití pro počítání počtu řešení, pravděpodobnostní sítě a eliminace minikošíků.

První část práce se bude zabývat teoretickým úvodem do Constraint processingu a optimalizačními problémy. V závěru první části se podíváme blíže na metody Eliminace košíku, které budeme používat v naší webové aplikaci.

Druhá část práce se bude věnovat samotné implementaci Eliminace košíku a popisu funkčnosti aplikace.

2 Úvod do Constraint processingu

Úlohy s omezenemími řešíme v běžném životě a obvykle při řešení těchto úloh nepotřebujeme počítač. Tyto úlohy člověk řeší intuitivně. Jedná se například o sestavování jídelníčku, sestavování rozvrhu atd. Většina úloh s omezeními je "výpočetně nezvládnutelná", nemůžeme tedy počítat s efektivními algoritmy řešící úlohy v plném rozsahu, ale můžeme navrhnout řešení efektivní pro většinu instancí. [1]

Základním stavebním kamenem je proměnná. Proměnná může nabývat hodnot, které má v doméně. Omezení jsou potom vztahy mezi proměnnými. Pro sestavování rozvrhu můžeme určit podmínky jako jsou například:

1. Vyučující nemůže vyučovat dva předměty zároveň
2. Student nemůže navštěvovat dva předměty zároveň
3. Učebna nesmí být obsazena

Podmínky lze reprezentovat:

1. Intenzionální (matematická/logická formule)
2. Extenzionální (výčet k-tic kompatibilních hodnot, 0-1 matice)

[2, 1]

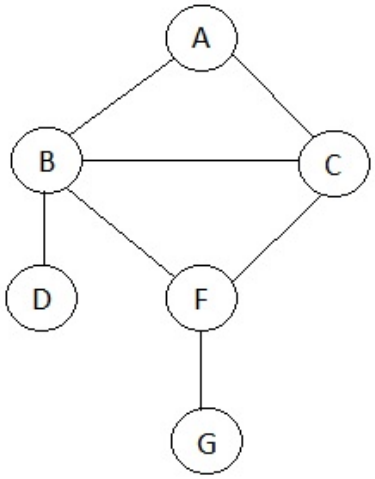
2.1 Síť omezení

Jedním z nejčastějších způsobů reprezentace úloh z oblasti constaint processingu jsou tzv. sítě omezení (Constraint network - CN). Každá síť omezení je tvořena z konečné množiny proměnných $X = \{x_1, ..., x_n\}$, z níchž každá proměnná má svou doménu $D_1, ..., D_n$ a množinu omezení $C_1, ..., C_t$. Každé omezení je vyjádřeno jako vztah definován na nějakou podskupinu proměnných. Omezení C je dáno dvěma parametry. Prvním je podmnožina proměnných $S_i = \{X_{i1}, ..., X_{ij}\}$ na kterých je omezení definováno (nazývá se scope neboli rozsah omezení). Druhým je relace definována na doméně proměnných z množiny $S_i : rel_i \subseteq D_{i1} \times ... \times D_{ij}$. Množina rozsahů je definována jako $S = \{S_1, ..., S_t\}$, toto se nazývá schéma sítě (network scheme). [4]

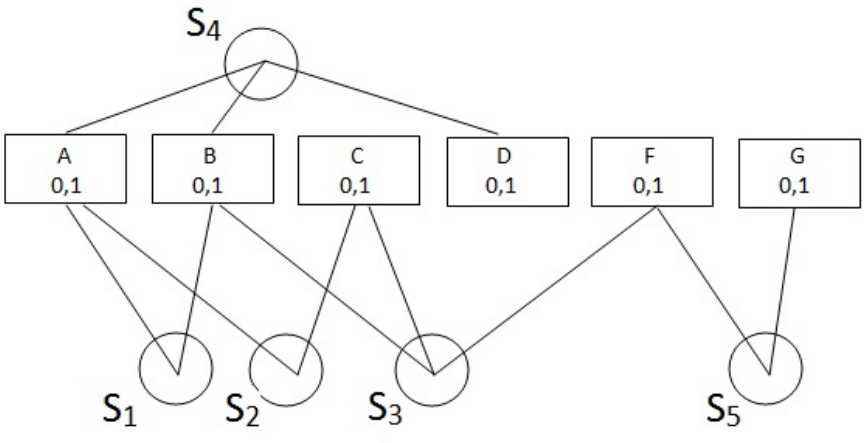
Constraint network lze reprezenovat pomocí grafu nebo hypergrafu. V obou případech každý vrchol reprezentuje proměnnou. Hrana mezi vrcholy grafu znamená, že existuje alespoň jedno omezení, v jehož rozsahu jsou obě příslušné proměnné současně. Každá hyperhrana v hypergrafu reprezentuje přesně jedno omezení - spojuje vrcholy odpovídající proměnným v rozsahu tohoto omezení. Příklad grafu vidíme na obrázku 1. Graf reprezentuje úlohu s množinou proměnných $X = \{a, b, c, d, f, g\}$ a omezení s rozsahy $S_1 = \{A, B\}$, $S_2 = \{A, C\}$, $S_3 = \{B, C, F\}$, $S_4 = \{A, B, D\}$ a $S_5 = \{F, G\}$.

Vidíme, že v grafu z hrany mezi vrcholy A,B nepoznáme, že tato dvojice proměnných se společně vyskytuje ve dvou různých omezeních. A také z grafu nepoznáme aritu omezení, ve kterých se proměnné vyskytují. Pro některé účely je však graf dostatečný, a proto se používá i přes tyto zmíněné nevýhody.

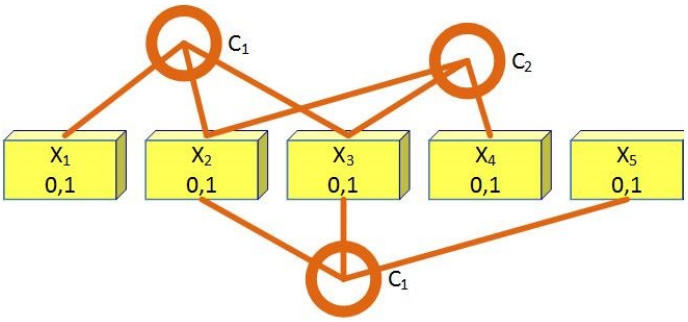
Pro porovnání si můžeme prohlédnout hypergraf vyobrazen na obrázku 2, který nám reprezentuje stejné zadání jako graf. Z hypergrafu můžeme vyčíst, že všechny proměnné mají doménu $\{0, 1\}$.[1]



Obrázek 1: Zobrazení CP pomocí grafu



Obrázek 2: Zobrazení CP pomocí hypergrafu



Obrázek 3: Hypergraf k příkladu

2.1.1 Ukázkový příklad

Pro ukázku vypočteme ukázkový příklad, jehož zadání zobrazuje i hypergraf na obrázku 3. Kompletní zadání tedy je:

- Zadané proměnné jsou: X_1, X_2, X_3, X_4, X_5
- Doména: {0,1} pro X_1, X_2, X_3, X_4, X_5
- Omezení jsou:
  - $C_1 : X_1 + X_2 + X_3 = 1$
  - $C_2 : X_1 - X_4 + X_5 > 0$
  - $C_3 : X_2 + X_3 - X_4 = 0$

Řešením je například $(X_1, X_2, X_3, X_4, X_5) = (1, 0, 0, 0, 1)$. Toto řešení splňuje všechny podmínky:

- $1 + 0 - 0 = 1$ - splněno
- $1 - 0 + 1 > 0$ - splněno
- $0 + 0 - 0 = 0$ - splněno

Řešením není například $(X_1, X_2, X_3, X_4, X_5) = (0, 0, 1, 1, 0)$. Tyto hodnoty nesplňují všechny podmínky:

- $0 + 0 - 1 = 1$ - nesplněno (vychází -1)
- $0 - 1 + 0 > 0$ - nesplněno (vychází -1)
- $0 + 1 - 1 = 0$ - splněno

3 Optimalizační problémy - Constraint Optimization

Problémy v reálném životě často zahrnují tvrdá (hard) a měkká (soft) omezení. Například v plánování rozvrhu omezení zdroje - učitel může učit pouze jednu třídu v jednom čase - musí být splněno. Zatímco žádost učitele učit pouze dva dny je jen preference, není to nezbytné. Když formalizujeme problémy, které mají měkká i tvrdá omezení, dostaneme síť omezení rozšířenou o globální cenovou funkci vyjadřující míru splnění měkkých omezení. V optimalizačních problémech hledáme taková přiřazení hodnot všem proměnným, která splňují všechna tvrdá omezení a optimalizují cenovou funkci.

Mnoho průmyslových problému je právě tohoto typu. Vezměme si například údržbu strojů v továrně na výrobu chladičů. Továrna má na výrobu určených 10 strojů. Je zapotřebí strojům naplánovat údržbu. Nemohou se zastavit všechny stroje najednou, protože továrna vyrábí 7 dní v týdnu 24 hodin v kuse, ovšem poptávka se mění. Na všech strojích nelze vyrábět všechny typy chladičů. Je tedy zapotřebí najít ve výrobě čas, kdy se jeden ze strojů může odstavit a může se provést údržba. Cílem plánu údržby je nezastavit výrobu, protože by firma mohla dostat sankce za nedodání materiálu, a určit posloupnost a délku trvání údržby. V tomto případě si jako tvrdé omezení můžeme stanovit, že na každý typ chladiče musí být alespoň jeden stroj, který ho vyrábí. Za měkké omezení lze zvolit nutnost odstávky co nejméně strojů. Každý by pak měl být odstaven co nejpozději v rámci jeho servisního intervalu, aby se minimalizoval počet servisů, a tedy i jejich cena.

Ovšem nejen v průmyslu nalézáme tento typ problémů. Lze je také najít v oblasti plánování, v umělé inteligenci, či v aukcích - nakonec každý úkol lze považovat za optimalizační problém.

Speciálním typem optimalizačních problémů jsou úlohy, kde jsou všechna omezení stejného typu. Snažíme se najít řešení, které tato omezení splňuje všechna a pokud to není možné, tak takové řešení, které maximalizuje počet splněných omezení. Takové úlohy označujeme jako Max-CSP. Příkladem může být Max-SAT (snaha najít ohodnocení formule v KNF splňující co nejvíce klauzulí). Omezením lze přiřadit váhu důležitosti. V takovém případě je úkolem, aby se minimalizoval součet vah porušených omezení. [1]

Optimalizační problém s omezeními (COP - constraint optimization problem) je možné reprezentovat sítí omezení rozšířenou o cenovou funkci. V kapitole 2.1 jsme popsali, co je síť omezení. Když k síti omezení přidáme množinu cenových komponent C_s, vznikne čtveřice $C = (X, D, C_h, C_s)$, kde (X, D, C_h) je síť omezení.

Cílem C_h je najít optimální řešení všech funkcí, tedy C_s je součet všech funkcí, které jsou co největší (maximální). [4]

4 Eliminace košíku pro optimalizaci - Bucket Elimination for Optimization

Popíšeme jeden z algoritmů pro řešení optimalizačních problému (constraint optimization). Pro ukázku uvažujme zadání:

- Proměnné $X = \{a, b, c, d\}$
- Omezení $C = \{F_0(a), F_1(a, b), F_2(a, d), F_3(b, d), F_4(a, b, c)\}$
- Cenová funkce $F(a, b, c, d) = F_0(a) + F_1(a, b) + F_2(a, d) + F_3(b, d) + F_4(a, b, c)$
- Pořadí proměnných $d_1 = a, b, d, c$

Tomuto zadání odpovídá graf 4. Domény a konkrétní specifikace funkcí nejsou pro popis algoritmu podstatné. Cílem je najít přiřazení hodnot proměnným, při kterém cenová funkce F má maximální hodnotu M. Tedy:

$$M =_{a,b,d,c}^{MAX} \{F_0(a) + F_1(a, b) + F_2(a, d) + F_3(b, d) + F_4(a, b, c)\}$$

Nyní budeme probírat proměnné v opačném pořadí, než je zvolené $d1$. Z maxima přes hodnoty probírané proměnné vytkneme funkce, které tuto proměnnou nemají mezi svými parametry. Tím, že hodnota těchto funkcí nezávisí na uvažované proměnné, není potřeba počítat maximum přes všechny možné hodnoty oné proměnné. V našem případě tedy jako první uvažujeme proměnnou c. Dostaneme:

$$M =_{a,b,d}^{MAX} \{F_0(a) + F_1(a, b) + F_2(a, d) + F_3(b, d) +_c^{MAX} \{F_4(a, b, c)\}\}$$

Pokračujeme dále s dalšími proměnnými. Nejprve s proměnnou d:

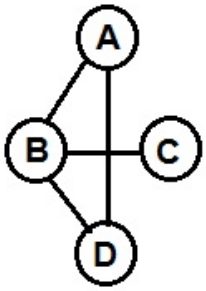
$$M =_{a,b}^{MAX} \{F_0(a) + F_1(a, b) +_d^{MAX} \{F_2(a, d) + F_3(b, d) +_c^{MAX} \{F_4(a, b, c)\}\}\}$$

Poté s proměnnou b:

$$M =_{a}^{MAX} \{F_0(a) +_b^{MAX} \{F_1(a, b) +_d^{MAX} \{F_2(a, d) + F_3(b, d) +_c^{MAX} \{F_4(a, b, c)\}\}\}\}$$

Pro poslední proměnnou není nikdy co vytknout (leda že by zde byly nějaké funkce bez parametrů), čili pro proměnnou a se nic nemění.

Pokud počítáme maximum přes jednu proměnnou ze součtu hodnot funkcí, které v parametrech mají proměnných více, nedokážeme výsledek vyjádřit číslem. Je možné jej ovšem vyjádřit jako funkci, která má za parametry všechny proměnné vyskytující se v tomto součtu, mimo proměnnou, přes kterou maximalizujeme. Opět pracujeme s proměnnými zprava doleva, tedy



Obrázek 4: CP pro demonstraci eliminace košíku

první budeme pracovat s proměnnou c. Vznikne nová funkce, která se značí $H^C(a,b)$. Ta je definována jako $H^C(a,b) =_c^{MAX} F_4(a,b,c)$. Novou funkci poté opět vytkneme mimo maximalizace přes proměnné, které nemá v parametrech. Po eliminaci proměnné c zbývají proměnné a a b. Maximalizace přes proměnnou d tedy hodnotu funkce $H^C(a,b)$ neovlivní, maximalizace přes b již ano. Získáme:

$$M =_a^{MAX} \{F_0(a) +_b^{MAX} \{F_1(a,b) + H^C(a,b) +_d^{MAX} \{F_2(a,d) + F_3(b,d)\}\}\}$$

Takto pokračujeme s dalšími proměnnými. Nyní budeme eliminovat proměnnou d. Vznikne nová funkce $H^D(a,b)$, která je definována jako $H^D(a,b) =_d^{MAX} \{F_2(a,d)+F_3(b,d)\}$. Tu umístíme k nejbližší proměnné. Po eliminaci proměnné d zbývají proměnné a a b, nejblíže je proměnná b dostáváme tedy:

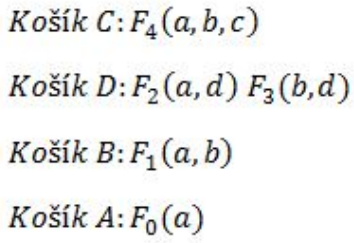
$$M =_a^{MAX} \{F_0(a) +_b^{MAX} \{F_1(a,b) + H^C(a,b) + H^D(a,b)\}\}$$

Nakonec eliminujeme proměnnou b. Vznikne nová funkce $H^B(a)$, která je definována jako $H^B(a) =_b^{MAX} F_1(a,b) + H^C(a,b) + H^D(a,b)$. Zbývá poslední proměnná a, tedy umístíme novou funkci k proměnné a. Získáme:

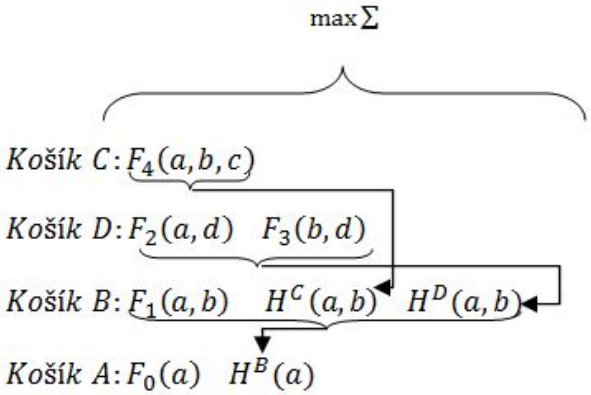
$$M =_a^{MAX} \{F_0(a) + H^B(a)\}$$

Výše popisovaný matematický postup zobrazuje rozdělování do organizační struktury nazývané košíky (bucket). Každý košík obsahuje sadu funkcí. Každá funkce bude v košíku té proměnné, která má nejvyšší pořadí ze všech proměnných, přes které danou funkci maximalizujeme (v matematickém zápisu to odpovídá maximalizaci uvedené těsně před nejvíce zanořeným párem závorek, ve kterých se funkce vyskytuje). Krok eliminace odpovídá příslušné funkci $H^x()$ a jejímu vytknutí mimo maximalizaci, která ji neovlivní (v košíku je to reprezentováno zařazením do příslušného košíku).

Rozdělení do košíků je závislé na zadaném pořadí $d_1 = a, b, d, c$. Košíky se tvoří v pořadí proměnných zprava doleva. Vzniknou tedy košíky C, D, B a A, do kterých rozřazujeme veškeré



Obrázek 5: Eiminace košíků - rozdělení funkcí



Obrázek 6: Eiminace košíků

funkce stejně jako ve výše uvedeném algoritmu. Na obrázku 5 je vyobrazeno rozdělení funkcí do košíků podle našeho ukázkového příkladu.

Po rozdělení funkcí se začíná eliminovat. Košíky se zpracovávájí od shora dolů. Prvně tedy zpracujeme košík C, z kterého nám vznikne nová funkce $H^C(a,b)$, a umístíme ji do nejbližšího košíku. Zbývají proměnné a a b. Nejblíže je košík B. Umístíme tedy $H^C(a,b)$ do košíku B. Takto pokračujeme se všemi košíky. Výsledek a postup eliminace je vyobrazen na obrázku 6.

Co se stane, když přehodíme pořadí proměnných? Pro příklad použijeme algoritmus uveden na obrázku 4. Proměnným určíte jiné pořadí $d_2 = a, d, b, c$. Prvně použijeme algebraickou manipulaci zprava doleva. Použitím d_2 získáme:

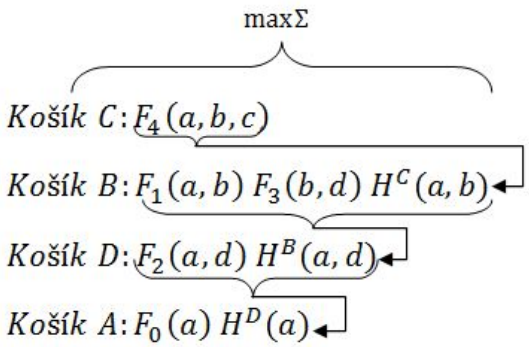
$$M =_{a,d,b,c}^{MAX} \{F_0(a) + F_1(a,b) + F_2(a,d) + F_3(b,d) + F_4(a,b,c)\} =$$

$$=_{a}^{MAX} \{F_0(a) +_{d}^{MAX} \{F_2(a,d) +_{b}^{MAX} \{F_1(a,b) + F_3(b,d) +_{c}^{MAX} \{F_4(a,b,c)\}\}\}\}$$

Po eliminaci vzniknout následující funkce:

$$H^C(a,b) =_{c}^{MAX} \{F_4(a,b,c)\}$$

$$H^B(a,d) =_{b}^{MAX} \{F_1(a,b) + F_3(b,d) + H^C(a,b)\}$$



Obrázek 7: Eiminace košíků s novým pořadím proměnných

$$H^D(a) =_d^{MAX} \{F_2(a,d) + H^B(a,d)\}$$

$$H^A =_a^{MAX} \{(F_0(a) + H^D(a)\}$$

Rozdělení do košíků s novým pořadím proměnných je vyobrazeno na obrázku 7. Z tohoto vyplývá, že při změně pořadí proměnných záleží, protože změní postup výpočtu (nezmění výsledné hodnoty).

Stejné výpočty budou provedeny použitím algoritmu ELIM-OPT, který je zobrazen na výpise 1. Z výše uvedeného výpočtu můžeme říci, že algoritmem ELIM-OPT je garantováno nalezení hodnoty optimálního řešení pomocí první fáze nazývané "zpětná"(protože jde od poslední proměnné k první). Druhá fáze "dopředná"(jde od první proměnné k poslední) poté nalezne konkrétní hodnoty všech proměnných, které tvoří optimální řešení všech košíkových funkcí, jež vedou k výběru maximálních hodnot ze všech proměnných. [1]

ELIM-OPT
Input: A cost network $C = (X, D, C)$, $C = \{F_1, ..., F_I\}$, $X = \{x_1, ...x_n\}$; ordering d.
Output: The maximal cost assignment to $\sum_j F_j$.
1. Initialize: Partition the cost components into ordered buckets.
2. Process buckets from p $\leftarrow$ n downto 1
For costs h_1, $h_2, \ldots, h_j$ defined over scopes $Q_1, .., Q_j$ in $bucket_{pi}$ do:
- If (observed variable) $x_p = a_{pi}$, assign x_p = a_{pi} to each h_i and put in appropriate buckets.
- Else (sum and maximize)
$A \leftarrow \cup_i Q_i - \{x_p\}$
$h^p = MAX_{xp} \sum_{i=1}^{j} h_i$
Place h^p in the latest lower bucket mentioning a variable in A.
3. Forward: From $i = 1$ to n, given $\vec{a}_{i-1}$, assign x_i a value a_i that maximizes the sum values of functions in its bucket.
4. Return the optimal cost computed in the bucket of x_1 and the otpimal assignment.

Výpis 1: Pseudokód ELIM-OPT[1]

4.1 Ukázka konkretního příkladu

Pro lepší pochopení výpočtu pomocí algoritmu eliminace košíku pro optimalizaci si spočteme ukázkový příklad s konkrétními hodnotami. Zadání:

- Zadané proměnné jsou : $X = \{a, b, c\}$
- Doména: $D = \{1, 2\}$ pro a, b, c
- Omezení: $C = \{F_0(b, c), F_1(a, b), F_2(a)\}$
- Cenová funkce: $F(a, b, c) = F_0(b, c) + F_1(a, b) + F_2(a)\}$
- Pořadí: $d = a, b, c$
- Hodnoty funkcí pro všechny hodnoty argumentů jsou vyobrazeny v tabulce 1

b	c	F_0
1	1	5
1	2	10
2	1	10
2	2	5

a	b	F_1
1	1	1
1	2	1
2	1	4
2	2	2

a	F_2
1	1
2	10

Tabulka 1: Zadání funkcí pro vzorový příklad - Eliminace košíku pro optimalizaci

b	H_c(b)
1	10
2	10

Tabulka 2: Funkce H_c vytvořena eliminací košíku C

a	b	F_1	H_c(b)	$\sum$
1	1	1	10	11
1	2	1	10	11
2	1	4	10	14
2	2	2	10	12

a	H_b(a)
1	11
2	14

Tabulka 3: Funkce H_b vytvořena eliminací košíku B

a	F_2	H_b(a)	$\sum$
1	1	11	12
2	10	14	24

Tabulka 4: Funkce H_a vytvořena eliminací košíku A

Nejprve můžeme použít algebraickou manipulaci zprava doleva s použitím pořadí d:

$$M =_{a,b,c}^{MAX} \{F_0(b,c) + F_1(a,b) + F_2(a)\} =_a^{MAX} \{F_2(a) +_b^{MAX} \{F_1(a,b) +_c^{MAX} \{F_0(b,c)\}\}\}$$

Výsledné košíky budou tedy vypadat takto:

$$Košík\ C : F_0(b,c)$$

$$Košík\ B : F_1(a,b) + H^C(b)$$

$$Košík\ A : F_2(a) + H^B(a)$$

Vytvoříme funkce z košíků. Jako první funkci vytvoříme nad košíkem C, tedy $H^C(b) =_c^{MAX} \{F_0(b,c)\}$. Konkrétní výpočet funkce $H^C(b)$ vidíme v tabulce 2, protože je zde pouze jedna funkce ($F_0(b,c)$) tak nesčítáme, ale pouze hledáme maximální hodnoty. Ve funkci $F_0(b,c)$, kterou si nalezneme v tabulce 1, hledáme maximální hodnotu pro b v každé její hodnotě. Tedy pro $b = 1$ porovnáváme hodnoty 5 a 10, tedy zapíšeme 10. Takto pokračujeme dále.

Další funkci z košíku vytvoříme nad košíkem B, tedy $H^B(a) =_b^{MAX} \{F_1(a,b) + H^C(b)\}$. Konkrétní výpočet se nám skládá ze dvou tabulek, které vídíme v tabulce 3. Zde se nám výpočet skládá ze dvou funkcí. Je tedy zapotřebí hodnoty prvně sečíst, a až poté hledáme maximální

hodnoty. Na tabulce vlevo vidíme součet hodnot, a na tabulce vpravo vidíme výsledné hodnoty (tedy maximální pro konkretní hodnoty).

Nakonec eliminujeme i poslední košík A, a dostaneme funkci $H^A =_a^{MAX} \{F_2(a) + H^B(a)\}$. Konkrétní výpočet je zobrazen na tabulce 4. Opět se nám skládá výpočet ze dvou funkcí. Tedy prvně funkce sečteme, a pak hledáme maximum. Maximum je hodnota 24, tedy když $a = 2$. Tímto jsme získali i řešení proměnné a, tedy proměnná $a = 2$.

Následuje další fáze výpočtu pro určení ostatních proměnných. Budeme dosazovat hodnoty již vyřešených proměnných do košíků odzadu. Košík A již máme vyřešen, takže vezmeme košík, který je před košíkem A, což je košík B. V košíku B máme dvě funkce: $F_1(a, b)+H^C(b)$, dosadíme všechny vyřešené proměnné - v našem případě za a dosadíme 2. Získáme $F_1(2, b) + H^C(b)$ a hledáme, kdy má b maximální hodnotu:

- Pro $b = 1$: $F_1(2, 1) + H^C(1) = 4 + 10 = 14$
- Pro $b = 2$: $F_1(2, 2) + H^C(2) = 2 + 10 = 12$

Z těchto výpočtu vidíme, že b je maximální v hodnotě 1. Tedy řešením je $b = 1$. Následuje výpočet další proměnné, a to pomocí košíku C. Opět hledáme, kdy má proměnná c maximální hodnotu. Košík C obsahuje jednu funkci $F_0(b, c)$, za proměnnou b dosadíme 1 (protože proměnná b je již vyřešena). Dostaneme tedy $F_0(1, c)$ a hledáme, kdy je proměnná c maximální:

- Pro $c = 1$: $F_0(1, 1) = 5$
- Pro $c = 2$: $F_0(1, 2) = 10$

Proměnná c je maximální v hodnotě 2, tedy řešením je $c = 2$. Protože již není žádný další košík, je výpočet hotov. Tato úloha má pouze jedno řešení - $(a, b, c) = (2, 1, 2)$.

5 Eliminace minikošíku

Stává se, že při eliminaci vznikne funkce s velkým počtem proměnných. Tímto se výpočet znesnadňuje a vyžaduje exponenciální prostor. Z tohoto důvodu vznikla v Constraint procesingu metoda eliminace na minikošíky. Košíky zde mohou být ještě děleny na menší části nazývané minikošíky. Díky tomu nemusí vzniknout při eliminaci košíku pouze jedna nová funkce, ale vzniká funkcí více z jednoho košíku (každá odpovídající jednomu minikošíku).

Při eliminaci košíku X s funkcemi $F_1(x, a)$, $F_2(x, a, b)$, $F_3(x, c, d)$ by vznikla pouze jedna funkce $H^x(a, b, c, d)$, ovšem při eliminaci minikošíků může vzniknout více funkcí z košíku X, tedy můžeme dostat například funkce $H^x(a, b)$ a $H^x(c, d)$.

Cenou za úsporu prostoru a času pro výpočet (místo počítání a ukládání hodnot pro všechny možné čveřice hodnot proměnných počítáme jen dvakrát hodnoty pro dvojice) je, že nemusíme získat optimální řešení. Získáme jen horní odhad nejlepšího řešení (protože maximalizujeme zvlášť funkci pro každý minikošík, mohou maxima vycházet větší díky použití různých hodnot pro x v obou případech, ale ve skutečném řešení je hodnota x jen jedna). A dále získáme nějaké řešení, které nemusí nutně mít maximální hodnotu cenové funkce (jeho hodnota je dolním odhadem na maximální řešení). Skutečné optimální řešení může mít hodnotu kdekoliv mezi tímto horním a dolním odhadem (včetně).

Metod rozdělování do minikošíků je více. Lze rozdělovat s pomocí dalšího parametru, kterému se říká stupeň rozkladu (kolik maximálně různých proměnných může být ve funkcích v jednom minikošíku). Né vždy je možné dosáhnout požadovaného stupně rozkladu, musí být splněná podmínka, že minimální stupeň rozkladu nesmí být menší než maximální počet proměnných ve funkci.

Když určíme stupeň rozkladu 5, eliminace může proběhnout následovně:

$$Košík A : F_0(a, b, c, d)F_1(a, b, e, f)$$

$$Košík B : H^A(b, c, d, e, f)$$

Ovšem, když určíme stupeň rozkladu 4, eliminace nemůže proběhnout stejným způsobem. Košík A musíme rozdělit na dva minikošíky, každý s jednou funkcí. Jejich eliminací vzniknou dvě nové funkce (první z minikošíku obsahujícího funkci $F0$ a druhá z minikošíku s funkcí $F1$):

$$Košík A : F_0(a, b, c, d)F_1(a, b, e, f)$$

$$Košík B : H^A(b, c, d)H^A(b, e, f)$$

Každá metoda rozdělování vede k jiné přesnosti.

My budeme eliminaci košíku provádět jiným způsobem. V momentě kdy v košíku eliminujeme budeme hledat všechny funkce podle proměnné, která je stejná jako název košíku.

Na obrázku 8 je zobrazeno porovnání rozdělování do košíku a minikošíku. Je zobrazen příklad z kapitoly 4 s pořadím $d_3 = a, b, c, d$.

Pro ukázku algoritmu vytvoříme algebraický zápis.

$$M =_{a,b,c,d}^{MAX} \{F_0(a) + F_1(a,b) + F_2(a,d) + F_3(b,d) + F_4(a,b,c)\}$$

Stejně jako u eliminace košíku pro optimalizaci budeme vybírat proměnné zprava doleva a hledat je ve funkcích. Získáme:

$$M =_a^{MAX} \{F_0(a) +_b^{MAX} \{F_1(a,b) +_c^{MAX} \{F_4(a,b,c) +_d^{MAX} \{F_2(a,d) + F_3(b,d)\}\}\}\}$$

Začneme eliminovat opět zprava doleva. V případě eliminace košíku pro optimalizaci by jsme eliminovali celý košík D. Ovšem u eliminace minikošíku nám vzniknou dvě nové funkce H^D. První je definována jako $H^D(b) =_d^{MAX} F_3(b,d)$, kterou umístíme do košíku B, a druhá je definovna jako $H^D(b) =_d^{MAX} F_2(a,d)$, kterou umístíme do košíku A. Získáme:

$$M =_a^{MAX} \{F_0(a) + H^D(a) +_b^{MAX} \{F_1(a,b) + H^D(b) +_c^{MAX} \{F_4(a,b,c)\}\}\}$$

Pokračujeme dále eliminaci košíku C. Vznikne nová funkce H^C, která je definována jako $H^C(a,b) =_c^{MAX} F_4(a,b,c)$ a kterou umístíme do košíku B. Získáme:

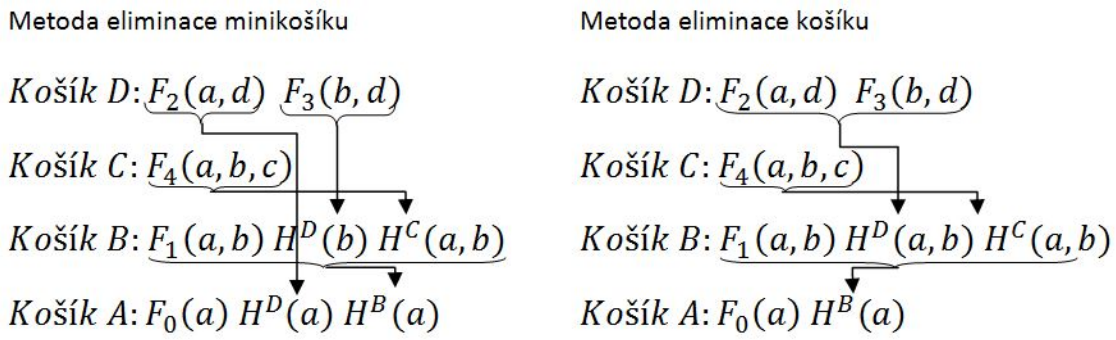
$$M =_a^{MAX} \{F_0(a) + H^D(a) +_b^{MAX} \{F_1(a,b) + H^D(b) + H^C(a,b)\}\}$$

Nakonec eliminujeme košík B. Vznikne nám nová funkce H^B, která je definována jako $H^B(a) =_b^{MAX} \{F_1(a,b) + H^D(b) + H^C(a,b)\}$ a kterou umístíme do posledního košíku A. Získáme:

$$M =_a^{MAX} \{F_0(a) + H^B(a)\}$$

Výpočet minikošíku je velmi podobný jako u Eliminace košíku pro optimalizaci. Zpětná fáze se počítá identicky.

Pseudokód k optimalizaci minikošíku nebo také MBE-OPT(i) můžeme vidět ve výpisu 2.[1]



Obrázek 8: Porovnání eliminace košíků a minikošíků

MBE-OPT(i)

Input: A cost network $C = (X, D, C)$; an ordering d; parameter i

Output: (a) An upper bound on the optimal cost solution, (b) an approximate solution, (c) a lower bound, and (d) the ordered augmented buckets.

1. Initialize: Partition the functions in C into $bucket_1, ..., bucket_n$ where $bucket_i$ constains all functions whose highest variable is x_i. Let $S_1, ..., S_m$ be the scopes of functions (new or old) in the processed bucket.

2.Backward For $p \leftarrow n$ downto 1, do

- If variable x_p is in instantiated $(x_p = a_p)$, assign $x_p = a_p$ to each h_i and put each resulting function into its appropriate bucket.
- Else, for $h_1, h_2, ..., h_m$ in $bucket_p$, generate an (i)-partitioning $Q' = \{Q_1, ..., Q_t\}$.

For each $Q_I \epsilon Q'$ containing $h_{I_1}, .., h_{I_t}$ generate function h^I, $h^I = MAX_{xp} \sum_{i=1}^{t} h_{I_i}$. Add h^I to the bucket of the largest-index variable in U_I, $U_I = \bigcup_{i=1}^{m}$ scope $(h_{I_i}) - \{x_p\}$.

3.Forward For $p = 1$ to n do, given $a_1, ..., a_{p-1}$ choose a value a_p of x_p that maximizes the sum of all the functions in x'_ps bucket.

4.Return the ordered set of augmented buckets, an assignment $\bar{a} = (a_1, ..., a_n)$, an interval bound (the value computed in $bucket_1$ and the cost $F(\bar{a})$).

Výpis 2: Pseudokód MBE-OPT[1]

6 Eliminace košíku pro použití počítání počtu řešení

Může se stát, že je zapotřebí spočítat počet řešení u úloh s tvrdými omezeními. Pro tento výpočet vznikla metoda eliminace košíku pro počítání počtu řešení, nazývána také jako ELIM-COUNT.

Algoritmus je podobný algoritmu eliminace košíku pro optimalizaci. Přiřazujeme hodnoty 1 nebo 0. 1 znamená konistentní řešení a 0 nekonzistentní řešení. Funkce vrací 1 pokud je funkce povolená entice, když je funkce nepovolená entice vrací 0. Tedy, aby bylo přiřazení hodnot proměnným konzistentní (a tedy šlo o řešení), je zapotřebí, aby všechny funkce reprezentující omezení vracely hodnotu 1. Když alespoň jedna funkce vrací hodnotu 0, je přiřazení hodnot proměnným nekonzistentní (tedy není řešením). Zda je přiřazení hodnot konzistentní tedy vypočteme násobením. Nakonec sečteme všechny konzistentní řešení a dostaneme požadovaný počet řešení. Oproti eliminaci košíku pro optimalizaci tedy vyměníme hledání maxima za sumu a sčítání funkcí v košíku za násobení.

Eliminace košíku pro počítání řešení lze využít i pro výpočet konkrétních řešení, a to za použití zpětné fáze. [1]

Pro algebraický zápis využijeme stejné zadání úlohy jako v kapitole 4 pro optimalizaci.

Počet řešení M počítáme:

$$M = \sum_{a,b,d,c} \{F_0(a) \cdot F_1(a,b) \cdot F_2(a,d) \cdot F_3(b,d) \cdot F_4(a,b,c)\}$$

Stejně jako u všech algoritmů budeme probírat proměnné v opačném pořadí, než je zvolené $d1$. Ovšem tentokrát nebudeme brát z maxima přes všechny hodnoty, ale ze sumy. Po úpravách tedy dostaneme:

$$M = \sum_a \{F_0(a) \cdot \sum_b \{F_1(a,b) \cdot \sum_d \{F_2(a,d) \cdot F_3(b,d) \cdot \sum_c \{F_4(a,b,c)\}\}\}\}$$

Po rozdělení funkcí začneme eliminovat opět stejně jako u algoritmu eliminace košíku pro optimalizaci. Nejprve budeme pracovat s proměnnou c. Vznikne nová funkce, která se značí $H^C(a,b)$, která je definována jako $H^C(a,b) = \sum_c F_4(a,b,c)$. Novou funkci opět vytkneme mimo sumu přes proměnné, které nemá v parametrech. Získáme:

$$M = \sum_a \{F_0(a) \cdot \sum_b \{F_1(a,b) \cdot H^C(a,b) \cdot \sum_d \{F_2(a,d) \cdot F_3(b,d)\}\}\}$$

Takto pokračujeme s dalšími proměnnými. Nyní budeme eliminovat proměnnou d. Vznikne nová funkce $H^D(a,b) = \sum_d \{F_2(a,d) \cdot F_3(b,d)\}$, kterou umístíme k nejbližší proměnné. Dostaneme:

$$M = \sum_a \{F_0(a) \cdot \sum_b \{F_1(a,b) \cdot H^C(a,b) \cdot H^D(a,b)\}\}$$

Eliminujeme další proměnnou - proměnnou b. Vznikne nová funkce $H^B(a) = \sum_b \{F_1(a,b) \cdot H^C(a,b) \cdot H^D(a,b)\}$. Získáme:

$$M = \sum_a \{F_0(a)\} \cdot H^B(a)\}$$

Na tomto algebraickém výpisu si můžeme snadno porovnat funkčnost algoritmu oproti eliminaci košíku pro optimalizaci.

Pseudokód je vyobrazen na výpisu kódu 3.[1]

ELIM-COUNT
Input: A costraint network $R = (X, D, C)$, ordering d;
Output: Augmented output buckets including the intermediate count functions and the total number of solutions.
1. Initialize: Partition C(0-1 cost functions) into ordered buckets $bucket_1, ..., bucket_n$.
We denote a function in a bucket N_i, and its scope S_i
2.Backward For $p \leftarrow n$ downto 1, do
Generate the function N^p : $N^p = \sum_{X_p} \prod_{N_i} \epsilon \text{bucket}_p N_i$.
Add N^p to the bucket of the latest variable in $\bigcup_{i=1}^{j} S_i - \{X_p\}$
3.Return the number of solutions, N^1 and the set of output buckets with the original and computed functions.

Výpis 3: Pseudokód ELIM-COUNT[1]

6.1 Ukázka konkrétního příkladu

Pro lepší pochopení algoritmu ELIM-COUNT si spočítáme vzorový příklad. Využijeme zadání hypergafu na obrázku 3 a zadání z kapitoly 2.1.1. Prvně si vytvoříme tabulky k omezením. Jedničkou označíme konzistentní řešení a nulou nekonzistentní. Výsledek vidíme v tabulce 5. Zadané pořadí je d=x_1, x_2, x_3, x_4, x_5 Rozdělíme si funkce do košíků:

$$Košík X_5 : F_2(x_1, x_4, x_5)$$

$$Košík X_4 : F_3(x_2, x_3, x_4) \cdot H^{X5}(x_1, x_4)$$

$$Košík X_3 : F_1(x_1, x_2, x_3) \cdot H^{X4}(x_1, x_2, x_3)$$

$$Košík X_2 : H^{X3}(x_1, x_2)$$

$$Košík X_1 : H^{X2}(x_1)$$

x_1	x_2	x_3	F_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

x_1	x_4	x_5	F_2
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

x_2	x_3	x_4	F_3
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Tabulka 5: Tabulky pro výpočet počtu řešení

Nyní si vypočteme funkce z košíku $H^{X_5}(x_1, x_4) = \sum_{X5} \{F_2(x_1, x_4, x_5)\}$. Výslednou tabulku vidíme v tabulce 6.

x_1	x_4	$H^{X5}(x_1, x_4)$
0	0	1
0	1	0
1	0	2
1	1	1

Tabulka 6: Tabulky výpočtu $H^{X5}(x_1, x_4)$

Pokračujeme na další funkci z košíku, která se vypočítává nad košíkem $X4$.

$$H^{X4}(x_1, x_2, x_3) = \sum_{X4} \{F_3(x_2, x_3, x_4) \cdot H^{X5}(x_1, x_4)\}$$

Výslednou tabulku vidíme v tabulce 7.

x_1	x_2	x_3	$h^{X4}(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	2
1	0	1	1
1	1	0	1
1	1	1	0

Tabulka 7: Tabulky výpočtu $H^{X4}(x_1, x_2, x_3)$

Pokračujeme na další funkci z košíku, která se vypočítává nad košíkem $X3$. $H^{X3}(x_1, x_2) = \sum_{X3} \{F_1(x_1, x_2, x_3) \cdot H^{X4}(x_1, x_2, x_3)\}$. Výslednou tabulku vidíme v tabulce 8.

x_1	x_2	$H^{X3}(x_1, x_2)$
0	0	0
0	1	0
1	0	2
1	1	0

Tabulka 8: Tabulky výpočtu $H^{X3}(x_1, x_2)$

Jako poslední nám zbývá eliminovat poslední košík. $H^{X2}(x_1) = \sum_{X2} \{H^{X3}(x_1, x_2)\}$. Výslednou tabulku vidíme v tabulce 9.

x_1	$H^{X2}(x_1)$
0	0
1	2

Tabulka 9: Tabulky výpočtu $H^{X2}(x_1)$

Výsledkem ukazuje, že počet řešení je 2. Když chceme zjistit konkrétní hodnoty řešení, použijeme zpětnou fázi. Tedy dosazujeme hodnoty proměnných do funkcí v košících. Řešením jsou všechny nenulové hodnoty. Bereme si poslední košík a zde vidíme, že pro proměnou x_1 je řešením hodnota 1. Pokračujeme na další košík:

$$X_3 = H^{X3}(x_1, x_2) = H^{X3}(1, 0) = 2$$

$$X_3 = H^{X3}(x_1, x_2) = H^{X3}(1, 1) = 0$$

Z tohoto výpočtu výplývá, že řešením je x_2 s hodnotou 0, protože nám zde vyšlo nenulové číslo, tedy číslo 2. Pokračujeme na další košík, tedy na X_3.

$$X_3 = H^{X4}(x_1, x_2, x_3) \cdot F_1(x_1, x_2, x_3) = H^{X4}(1, 0, 0) \cdot F_1(1, 0, 0) = 2 \cdot 1 = 2$$

$$X_3 = H^{X4}(x_1, x_2, x_3) \cdot F_1(x_1, x_2, x_3) = H^{X4}(1, 0, 1) \cdot F_1(1, 0, 1) = 1 \cdot 0 = 0$$

Z tohoto výpočtu vyplývá, že řešením je x_3 s hodnotou 0, protože zde vyšlo nenulové číslo, a to číslo 2. Pokračujeme dále na košík X_4.

$$X_4 = H^{X5}(x_1, x_4) \cdot F_3(x_2, x_3, x_4) = H^{X5}(1, 0) \cdot F_3(0, 0, 0) = 1 \cdot 2 = 2$$

$$X_4 = H^{X5}(x_1, x_4) \cdot F_3(x_2, x_3, x_4) = H^{X5}(1, 1) \cdot F_3(0, 0, 1) = 0 \cdot 1 = 0$$

Z tohoto výpočtu vyplývá, že řešením je x_4 s hodnotou 0. Pokračujeme na poslední košík X_5.

$$X_5 = F_2(x_1, x_4, x_5) = F_2(1, 0, 0) = 1$$

$$X_5 = F_2(x_1, x_4, x_5) = F_2(1, 0, 1) = 1$$

Z tohoto výpočtu nám výplývá, že x_5 má řešení s hodnotou 0 i 1. Řešením tedy je $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 0, 0, 0)a(x_1, x_2, x_3, x_4, x_5) = (1, 0, 0, 0, 1)$. Při dosazení hodnot do zadaných rovnic vidíme, že to opravdu funguje. Ověření pro řešení $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 0, 0, 0)$:

- F_1: $X_1 + X_2 + X_3 = 1 \Rightarrow F_1 : 1 + 0 + 0 = 1 \Rightarrow Platí$
- F_2:X_1 - $X_4 + X_5 > 0 \Rightarrow F_2 : 1 - 0 + 0 > 0 \Rightarrow Platí$
- F_3:$X_2 + X_3$ - $X_4 = 0 \Rightarrow F_3 : 0 + 0 - 0 = 0 \Rightarrow Platí$

Všechny 3 podmínky byly splněny. Zkusíme si to i pro druhé řešení, tedy pro $(x_1, x_2, x_3, x_4, x_5) = (1, 0, 0, 0, 1)$:

- F_1: $X_1 + X_2 + X_3 = 1 \Rightarrow F_1 : 1 + 0 + 0 = 1 \Rightarrow Platí$
- F_2:X_1 - $X_4 + X_5 > 0 \Rightarrow F_2 : 1 - 0 + 1 > 0 \Rightarrow Platí$
- F_3:$X_2 + X_3$ - $X_4 = 0 \Rightarrow F_3 : 0 + 0 - 0 = 0 \Rightarrow Platí$

Všechny 3 podmínky byly opět splněny.

7 Pravděpodobnostní sítě - Probabilistic Networks

V některých případech může být vztah mezi proměnnými určen pravděpodobností. Patří zde například hodnocení situace, lékařská diagnostika, pravděpodobnostní dekódování atd. Tyto úlohy reprezentujeme pravděpodobnostní sítí.[1]

Pravděpodobnostní síť (Probabilistic network) se také nazývá jako Belief network nebo Bayesian network. Pravděpodobnostní síť je definována jako acyklický orientovaný graf, kde uzly reprezentují proměnné a hrany vliv mezi proměnnými. Pro příklad můžeme uvést lékařské diagnostiky. Více nemocí může mít různé symptomy, stejná nemoc u různých lidí může mít různé příznaky. Některé příznaky mají všichni a některé příznaky nemoc provázejí méně často. [6]

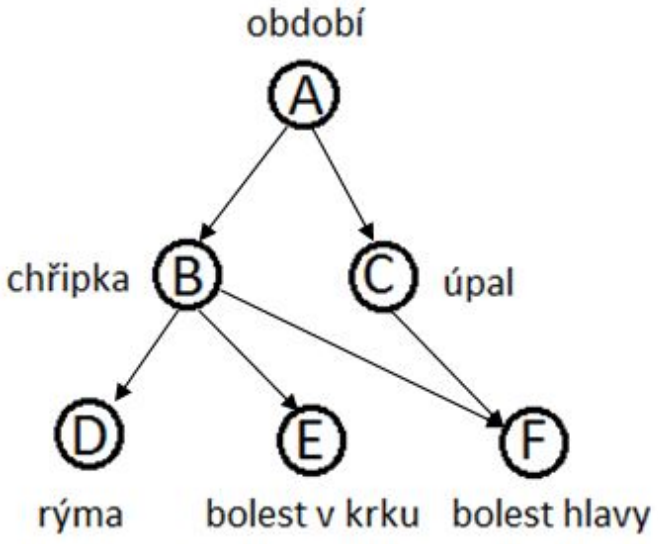
Pro hloubější přehled se podívejte do této knihy [6], a úplné zpracování belief networks je uvedeno v knihách [7, 8].

Příklad pravděpodobnostní sítě je zobrazen na obrázku 9. Tímto grafem můžeme vyjádřit různé vztahy například mezi obdobími (A) a nemocemi (B) a (C). Z grafu lze také vyčíst, že neexistuje přímá souvislost mezi (A) a (F).

Definice Belief network: mějme množinu proměnných $X = \{x_1, ..., x_n\}$ nad doménami $D_{x_1}, ..., D_{x_n}$. Belief network tvoří pár $\langle G, P \rangle$, kde $G = (X, E)$ je orientovaný acyklický graf nad proměnnými a $P = \{P_i\}$, která je podmíněná pravděpodobnost tabulky $P_i = \{P(x_i|pa_i)\}$. Tabulku P_i nazýváme CPT. x_i nám udává proměnnou, jež je pravděpodobnostně závislá na proměnných pa_i. pa_i je tedy množina proměnných, na kterých je závislá proměnná x_i. [1]

Algoritmů pro využití pravděpodobnostních síti je více. Patří sem:

- Belief assessment - pozorováním jsou zjištěné hodnoty některých proměnných, cílem je dopočítat pravděpodobnost pro každou možnou hodnotu každé proměnné pro dané pozorování
- MPE - the most probable explanation - při daném pozorování hledáme pro všechny ostatní proměnné hodnotu z jejich domény, která má v té situaci nějvětší pravděpodobnost
- MPA - the maximum a posteriori hypothesis - hledání přiřazení hodnot podmnožině nepozorovaných proměnných, které maximalizuje jejich podmíněnou pravděpodobnost



Obrázek 9: Graf pravděpodobnostní sítě

Obrázek 10: Graf pravděpodobnostní sítě

7.1 Eliminační algoritmus pro MPE

Pro implementaci jsme si vybrali algoritmus MPE, tedy pro nalezení nejpravděpodobnějšího vysvětlení. Vstupní hodnoty jsou vždy v intervalu od 0 do 1.

Pro ukázku funkčnosti algoritmu mějme jednoduché zadání:

- Pravděpodobnostní síť $P = \{P_0(c|b), P_1(b|a), P_2(a)\}$
- Proměnné $X = \{a, b, c\}$
- Pořadí proměnných $d_1 = a, b, c$
- Graf pravděpodobnostní sítě je vyobrazen na obrázku 10

Podle definice pak počítáme:

$$M =_{a,b,c}^{MAX} \{P_0(c|b) \cdot P_1(b|a) \cdot P_2(a)\}$$

Nyní budeme probírat proměnné v opačném pořadí, než je zvolené d_1. V našem případě prvně uvažujeme proměnnou c. Nyní je pro nás důležitá proměnná před symbolem $|$, kterou vytkneme před maximum. Dostaneme:

$$M =_{a,b}^{MAX} \{P_1(b|a) \cdot P_2(a) \cdot_{c}^{MAX} \{P_0(c|b)\}\}$$

Pokračujeme opět dále s dalšími proměnnými. Nejprve s proměnnou b:

$$M =_{a}^{MAX} \{P_2(a) \cdot_{b}^{MAX} \{P_1(b|a) \cdot_{c}^{MAX} \{P_0(c|b)\}\}\}$$

Nakonec pracujeme s poslední proměnnou a, kde nám zbyla poslední CPT, a tedy se nic nemění:

$$M =_{a}^{MAX} \{P_2(a) \cdot_{b}^{MAX} \{P_1(b|a) \cdot_{c}^{MAX} \{P_0(c|b)\}\}\}$$

Podobně jako u algoritmu eliminace košíku pro optimalizaci, vyjádříme maximum novou funkcí. Opět pracujeme s proměnnými zprava doleva, tedy první budeme pracovat s proměnnou c. Vznikne nová funkce, která se značí $\lambda_C(b)$ (Zde vidíme rozdíl oproti eliminace košíku pro optimalizaci, kde se nová funkce značila velkým písmenem H), která je definována jako $\lambda_C(b) =_{c}^{MAX} \{P_0(c|b)\}$. Novou funkci opět vytkneme mimo maximalizace přes proměnné, které nemá v parametrech. Získáme:

$$M =_{a}^{MAX} \{P_2(a) \cdot_{b}^{MAX} \{P_1(b|a) \cdot \lambda_C(b)\}\}$$

Takto pokračujeme s dalšími proměnnými. Nyní budeme eliminovat proměnnou b. Vznikne nová funkce $\lambda_B(a)$, která je definována jako $\lambda_B(a) =_{b}^{MAX} \{P_1(b|a) \cdot \lambda_C(b)\}$. Dostáváme tedy:

$$M =_{a}^{MAX} \{P_2(a) \cdot \lambda_B(a)\}$$

Nyní si opět můžete algoritmus porovnat s algortimem eliminace košíku pro optimalizaci uvedeného v kapitole 4. Hlavním rozdílem je, že místo sčítání zde násobíme a nová funkce po eliminaci se neznačí velkým písmenem H, ale symbolem $\lambda_X(y_1, , .., y_n)$, kde X označuje z kterého košíku funkce vznikla, a $y_1, ..y_n$ jsou proměnné, které zbyly po eliminaci.

Pseudokód algoritmu ELIM-MPE si můžeme prohlédnout na výpisu kódu 4.[1]

ELIM-MPE

Input: A belief network $\langle G, P\rangle, P = \{P_1, .., P_n\}$; an ordering of the variables, d; observations e.

Output: The most probable assignment

1. Initialize: as in ELIM-BEL
2. Backward : For $p \leftarrow n$ downto 1, do for all the functions $h_1, h_2, .., h_j$ in $bucket_p$, do
   - if (observed variable) $bucket_p$ constains $x_p = a_p$, assign $x_p = a_p$ to each h_i and put each in appropriate bucket.
   - else, $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{x_p\}$. Generate functions $h_p = MAX_{x_p} \prod_{i-1}^{j} h_i$. Add h_p to bucket of largest-index variable bucket in U_p.
3. Return: The mpe value is obtained by the constant computed in $bucket_1$.

An mpe tuple is obtained by assigning values in the ordering d consulting recorder functions in each bucket as follows. Given the assignment $a = a_1, .., a_{i-1}$ choose $a_i = argmax_{x_i} \prod_{\{h_j \epsilon bucket_i | a=(a_1,..,ai-1\}} h_j$

Výpis 4: Pseudokód ELIM-MPE[1]

7.2 Ukázka konkretního příkladu

Pro praktické vysvětlení si vypočteme konkrétní příklad. Použijeme stejné zadání z kapitoly 7.1, které si upřesníme o pravděpodobnostní tabulky:

- Pravděpodobnostní síť $P = (a, b, c) = P_0(c|b)P_1(b|a)P_2(a)$
- Pořadí proměnných $d_1 = a, b, c$
- Graf pravděpodobnostní sítě je vyobrazen na obrázku 10
- Pravděpodobnostní tabulky viz Tabulka 10

b	c	$P_0(c\|b)$
1	1	0.3
2	1	0.1
1	2	0.2
2	2	0.4

a	b	$P_1(b\|a)$
1	1	0.2
2	1	0.2
1	2	0.4
2	2	0.2

a	$P_2(a)$
1	0.3
2	0.7

Tabulka 10: Pravděpodobnostní tabulky

Nejprve si CPT rozdělíme do košíku a eliminujeme. Vznikne (viz. kapitola 7.1):

$$KošíkC : P_0(c|b)$$

$$KošíkB : P_1(b|a)\lambda_C(b)$$

$$KošíkA : P_2(a)\lambda_B(a)$$

Nyní vypočteme nově vzniklé funkce. Nejprve nad košíkem C, tedy $\lambda_C(b) =_c^{MAX} \{P_0(c|b)\}$, protože je zde pouze jedna funkce, tak hodnoty nenásobíme, ale pouze hledáme maximální hodnoty. V CPT P_0, kterou nalezneme v tabulce 10, hledáme maximální hodnotu pro b v každé její hodnotě - tedy pro $b = 1$ porovnáváme hodnoty 0.3 a 0.2. Zapíšeme tedy hodnotu 0.3. Výslednou tabulku funkce $\lambda_C(b)$ vidíme v tabulce 11.

b	λ_C
1	0.3
2	0.4

Tabulka 11: Tabulka výpočtu $\lambda_C(b)$

Pokračujeme dalším košíkem, a to košíkem B. Vznikne nám funkce $\lambda_B(a) =_b^{MAX} \{P_1(b|a) \cdot \lambda_C(b)\}$. Nyní jsou v košíku již dvě funkce, a proto se výpočet skládá ze dvou tabulek. Zapotřebí je tedy hodnoty prvně násobit, a poté hledat maximální hodnoty. Pomocná tabulka pro výpočet

je umístěna vlevo a výsledná tabulka je umístěna vpravo v tabulce 12. Nejprve tedy hodnoty násobíme, a poté hledáme maxima pro stejnou hodnotu proměnné a.

a	b	$P_1(b\|a)$	$\lambda_C(b)$	*Součin*
1	1	0.2	0.3	0.06
2	1	0.2	0.3	0.06
1	2	0.4	0.4	0.16
2	2	0.2	0.4	0.08

a	$\lambda_B(a)$
1	0.16
2	0.08

Tabulka 12: Tabulka výpočtu $\lambda_B(a)$

Nakonec eliminujeme i poslední košík A. Dostaneme funkci $\lambda_A()$, která se opět skládá ze dvou funkcí. Počítáme ji stejným způsobem jako v předchozím případu. Funkce prvně vynásobíme, a následně hledáme maximum. Vypočítáme hodnoty pro $a = 1$, v tabulce 10 si najdeme tabulku pro $P_0(a)$ a hodnotu pro $a = 1$. Zde je hodnota 0.3, kterou vynásobíme s hodnotou z tabulky 12 pro $a = 1$, která je 0.16. Součinem hodnot 0.3 a 0.16 dostáváme výsledek 0.048. Shodně postupujeme pro $a = 2$, kde dostaneme výsledek 0,056. Po porovnání je maximum, když $a = 2$. Takto jsme získali i řešení proměnné a - proměnná $a = 2$.

Následuje stejně jako u jiných algoritmů další fáze výpočtu pro určení konkrétních hodnot ostatních proměnných. Budeme dosazovat hodnoty již vyřešených proměnných do košíku odzadu. Pokračujeme košíkem B. Za proměnnou a dosadíme hodnotu 2. V košíku B máme $P_1(b|a)$ a $\lambda_C(b)$. Dopočítáme tedy hodnoty pro všechny hodnoty, které b může nabývat:

- Pro $b = 1 : P_1(1|2) \cdot \lambda_C(1) = 0.2 \cdot 0.3 = 0.06$
- Pro $b = 2 : P_1(2|2) \cdot \lambda_C(2) = 0.2 \cdot 0.4 = 0.8$

Z těchto výpočtu vidíme, že b je maximální v hodnotě 2. Tedy řešením je $b = 2$. Pokračujeme na poslední košík, košík C. Opět počítáme stejným způsobem:

- Pro $c = 1 : P_0(1|2) = 0.1$
- Pro $c = 2 : P_0(2|2) = 0.4$

Z toho vyplývá, že proměnná c je maximální v hodnotě 2, řešením je $c = 2$. Propočtem posledního košíku je výpočet hotov. Tato úloha má pouze jedno řešení - $(a, b, c) = (2, 2, 2)$.

8 Analýza požadavků - funkční a nefunkční požadavky

8.1 Funkční požadavky

- **PROČ?** Proč tento systém vznikl? Systém vznikl pro podporu výuky ARUO, aby studenti lépe pochopili algoritmy z oblasti Constraint processingu, konkrétně metody Eliminace košíku.
- **K ČEMU?** K čemu tento systém bude sloužit? Systém bude sloužit k zobrazení každého kroku výpočtu určitého algoritmu, a pomůže studentům (uživatelům) pochopit na zobrazeném postupu funkčnost algoritmů na svých vstupech či na vzorových příkladech.
- **KDO?** Kdo bude systém využívat? Systém je zaměřen na studenty předmětu ARUO, nebo na kohokoliv, koho zajímá Constraint processing a metoda Eliminace košíku.
- **VSTUPY A VÝSTUPY** Vstupy systému jsou zadání pro výpočet metody eliminace košíku (V našem případě buď síť omezení nebo pravděpodobnostní síť). Výstupy jsou pak jednotlivé kroky a výpočet problému.

8.2 Nefunkční požadavky

Systém je omezen na zadání počtu proměnných a funkcí, aby výpočet netrval nepřiměřeně dlouho.

Systém je odzkoušen na prohlížečích Safari, Internet Explorer, Opera a Google Chrome. Není testován na prohlížečích jako jsou například prohlížeč od firmy Seznam, Pampa, Maxthon atd. Systém byl také zkoušen na mobilním telefonu Iphone 6 a Samsung galaxy S5 a také na iPadu mini 4. Není testován na starších mobilních zařízeních.

9 Implementace

Implementace výukového serveru pro ARUO je realizována jako webová aplikace v jazyce Java spouštěná pomocí serveru Apache Tomcat v7.0. Výsledný projekt je složen z části pro web, převážně stránky jsp všetně css stylu a javascriptu (klientská část) a s části programu pro výpočty, který běží na serveru a vrací zpět klientovi výsledky pomocí uživatelského rozhraní webové stránky. Pro software na serveru je v příloze na CD vytvořena dokumentace javadoc.

9.1 Využívané objekty

V javovském package „eu.aruo.data“ se nachází objekty, se kterými pracujeme. Jejich třídní diagram si můžeme prohlédnout na obrázcích 11 a 12. Objekt `Bucket` představuje košík. Má parametry id, název, proměnné, funkce, funkce z košíku, a ve kterém košíku je použit. Obsahuje 2 konstruktory: jeden bez parametrů a jeden s parametry, které jsme před chvíli jmenovali. Obdobně jsou vytvořeny i další objekty. Podle názvu lze lehce rozeznat, který objekt co představuje.

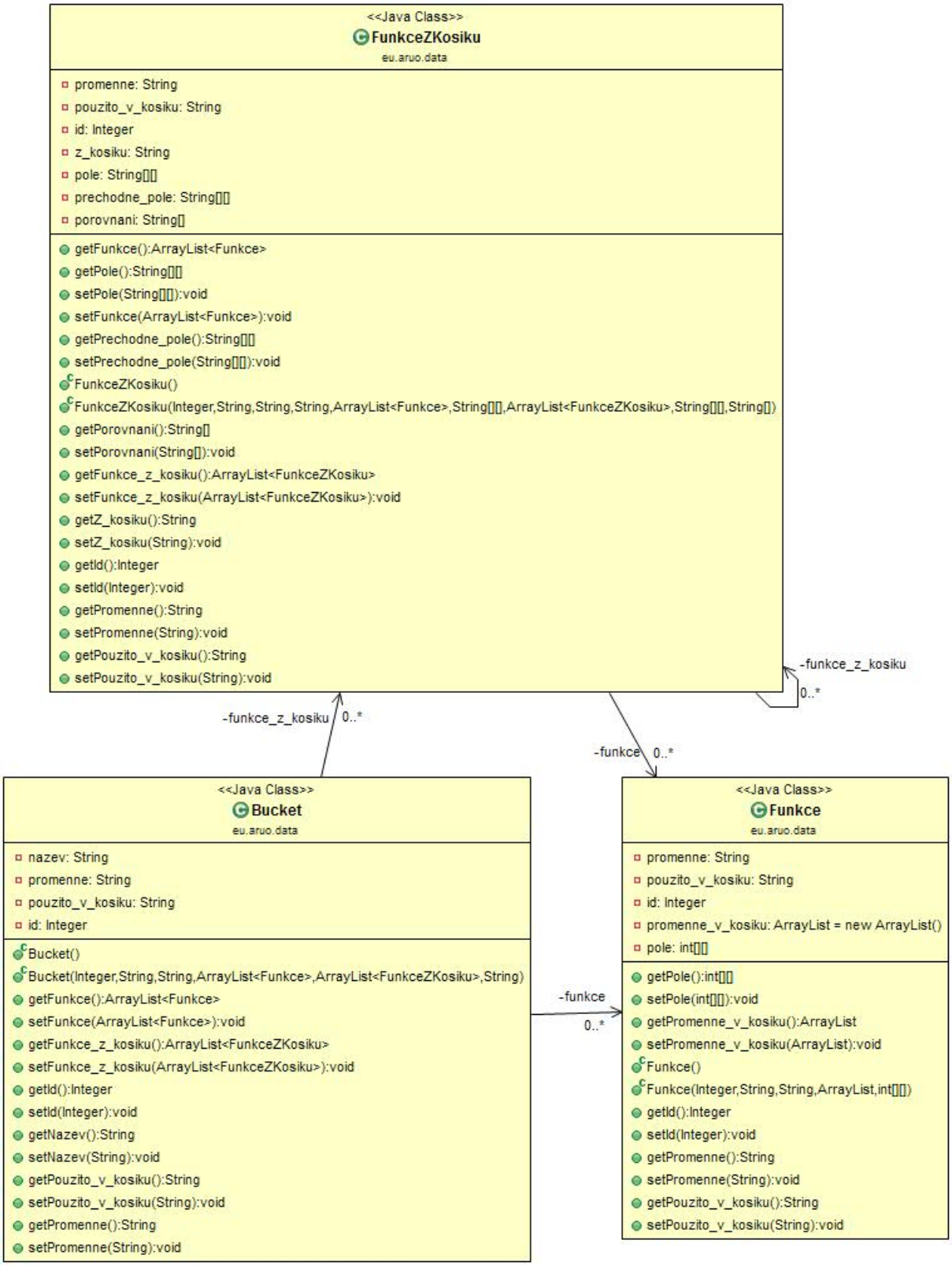
Tyto objekty využiváme pro metody Eliminace košíku pro optimalizaci, pro zjištění počtu řešení a pro eliminaci minikošíku. Pro Eliminaci košíku a jeho metodu s využitím pravděpodobnostních sítí jsou vytvořeny podobné objekty v javovské package „eu.aruo.pravdepodobnost_data“. Jediným rozdílem jsou zde jiné datové typy, které využívají pouze pravděpodobnostní sítě, tedy místo s integery počítáme s datovým typem double.

9.2 Eliminace košíku pro optimalizaci - část SW na serveru

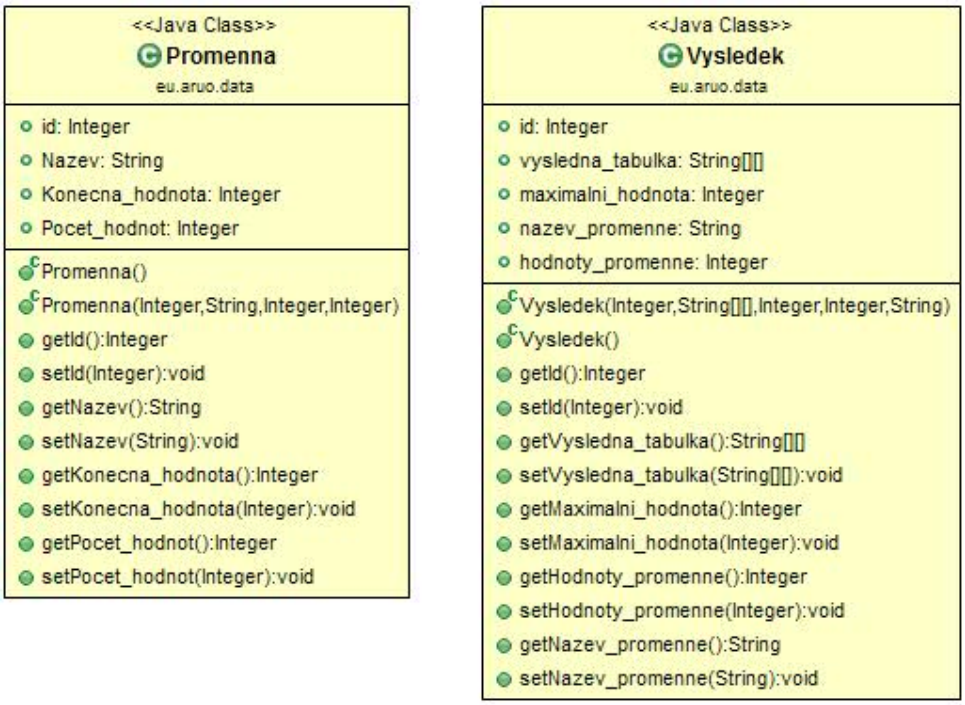
Třída `OptimalizaceServlet` se nachází v java package „eu.aruo.program“ a obsahuje všechny výpočty Eliminace košíku pro optimalizaci. Nejprve se do třídy nahrají všechny prvky zadané na webové stránce, se kterými poté počítáme. Protože webová stránka posílá všechny proměnné typu String, je nutné je správně přetypovat. Poté se prvky z webové stránky nahrají do požadovaných objektů. Ukázku tohoto úkonu vidíme na výpisu zdrojového kódu 5, kde lze dobře pozorovat načtení `funkce_konce` a jeho následné přetypování ze stringu na integer. Nakonec se vytvoří objekt `Funkce`, který se vloží do listu, kde ukládáme všechny funkce, s kterými chceme pracovat. Takto dostaneme všechna potřebná data ze stránky a vložíme si je do programu, který běží na serveru.

Když máme všechna data připravena, můžeme začít řešit výpočet. Nejprve je zapotřebí funkce rozdělit do košíků a vytvořit funkce z košíku, které jsou v košících také umístěny. K tomuto slouží metoda `rozrazeni_do_kosiku`. Tato metoda nejprve vytvoří prázdné košíky podle zadaného pořadí, poté rozdělí funkce do košíků a nakonec vytvoří funkce z košíků, které také rozdělí do správných košíků.

Po rozdělení všech funkcí a funkcí vzniklých z košíku do správných košíků: následuje metoda `uprava_promennych`. Úprava proměnných eliminuje opakující písmena či mezery vzniklé rozdělováním. Využívá se zde práce ze stringy, jejich procházení a úprava.



Obrázek 11: Úryvek třídního diagramu



Obrázek 12: Zbývající objekty

Obrázek 13: Třída OptimalizaceServlet

Když jsou košíky připraveny a upraveny, přichází na řadu samotný výpočet funkcí vzniklých z košíků. K tomuto slouží metoda `vypocet_tabulek`, kde se nám vytvoří tabulky s výpočty funkcí z košíků. Metoda si nejprve vytvoří přechodnou tabulku na konkrétní výpočet, a poté výslednou tabulku, kde ukládá všechny vypočtené hodnoty. Tyto hodnoty se poté zobrazují uživateli s mírným popisem postupu. Na tuto metodu navazuje metoda `nove_zobrazeni`, která byla doplněna později, a slouží k zobrazení postupu výpočtu ve výsledné tabulce. Tímto skončila první fáze.

Po první fázi začneme počítat druhou fázi, abychom zjistili konkrétní hodnoty proměnných. Pro tento výpočet nám slouží metoda `zpetny_vypocet`. Prvně vypočteme eliminací posledního košíku první proměnnou. Další proměnné počítáme s již vypočtenou proměnnou. Výpočet probíhá výpočtem funkcí a funkcí z košíku v košíku a vytvořením konečných hodnot proměnných. Pokud by nastala situace s více než jedním řešením, tak v tomto okamžiku se výsledků vytváří víc s nedopočtenými dalšími proměnnými. Pro jejich dopočet pak slouží metoda `dodatecny_vypocet`.

Po výpočtu se zkontroluje, zda se náhodou nějaký výsledek neopakuje. Všechna vypočtená data se odešlou na zobrazení do webové stránky a otevře se následující webovou stránka `optimalizace4.jsp`. Tento úkon vidíme na výpisu zdrojového kódu 6.

```
for (int i = 0; i < pocet_funkci; i++) {
   String funkce_kon = request.getParameter("funkce_konce" + i);
   int funkce_radky = Integer.parseInt(funkce_kon);
   String funkce_slou = request.getParameter("funkce_sloupce" + i);
   int funkce_sloupce = Integer.parseInt(funkce_slou);
   String pole[][] = new String[funkce_radky + 1][funkce_sloupce + 1];
   int pole2[][] = new int[funkce_radky + 1][funkce_sloupce + 1];
   String promenne = "";
   for (int j = 0; j <= funkce_radky; j++) {
      for (int k = 0; k <= funkce_sloupce; k++) {
         if (j == 0 && k != funkce_sloupce) {
            promenne = promenne + request.getParameter("funkce" + i + "radek" +
                j + "sloupec" + k);
         }
      pole[j][k] = request.getParameter("funkce" + i + "radek" + j + "sloupec"
          + k);
      }
   }
   for (int j = 1; j <= funkce_radky; j++) {
      for (int k = 0; k <= funkce_sloupce; k++) {
         if (j == 1) {
            pole2[0][k] = 0;
         }
         pole2[j][k] = Integer.parseInt(pole[j][k]);
      }
   }
   ArrayList<String> list_promenne = new ArrayList<String>();
   for (int i1 = 0; i1 < promenne.length(); i1++) {
      list_promenne.add(promenne.charAt(i1) + "");
   }
   list.add(new Funkce(i, promenne, null, list_promenne, pole2));
}
```

Výpis 5: Ukázka načtení dat ze stránky a zařazení ke správnému objektu

```
request.getSession().setAttribute("vysledek_cely", listy_vysledku);
request.getSession().setAttribute("kosiky_list", bucket);
getServletContext().getRequestDispatcher("/optimalizace4.jsp").forward(request,
    resp);
eu.aruo.data.data.kosiky = bucket;
eu.aruo.data.data.vysledek = listy_vysledku;
for (int j = 0; j < vysledek.size(); j++) {
   request.getSession().setAttribute("vysledek_list" + j,
   vysledek.get(j).getNazev_promenne() + " " + vysledek.get(j).
       getHodnoty_promenne());
}
```

Výpis 6: Ukázka poslání dat na web

9.3 Eliminace košíku pro optimalizaci - webová část

Webová část softwaru je tvořena pomocí JSP a CSS stylů. Pro eliminaci košíku využiváme JSP vyobrazeny na obrázku 14. `optimalizace.jsp` až `optimalizace7.jsp` slouží pro klasický výpočet bez pevně zadaného vstupu. Pro vzorové příklady slouží JSP s názvem obsahující slovo „zadane“, příklady s nezměnitelným vstupem se můžou přídavat podle potřeb vyučujícího. Zadané příklady obsahují pouze po třech JSP, protože dále je již vše řešeno klasickým způsobem.

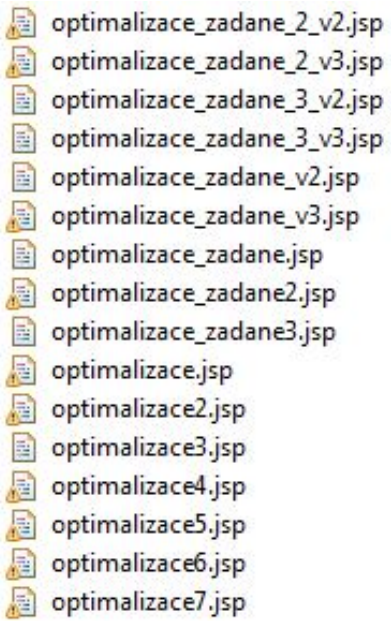
Při testování webových prohlížečů jsme nalezli velký rozdíl mezi prohlížečem Firefox a Safari. U prohlížeče Firefox nefungovaly klávesy na smazání a tabulátor pro zadávání hodnot. Zjistili jsme, že tomuto prohlížeči vadí událost `onkeypress`, kterou se mělo omezenit vkládání nepovolených znaků. Po úpravě zadávacích polí zase nefungovalo vše správně u prohlížeče Safari. Nakonec jsem tedy zvolila variantu vyfiltrovat si prohlížeč Safari a v něm pole pro vyplnění vytvářet jiným způsobem. Pomocí `UserAgenta` jsem si zjistila, jaký prohlížeč uživatel právě používá, a pokud to je prohlížeč Safari, tak se stránka vysází jiným způsobem. Příklad využití `UserAgenta` je zobrazen na výpisu zdrojového kódu 7. Po zadání všech potřebných hodnot se hodnoty odešlou na server ke zpracování.

Vstupní hodnoty z klávesnice také ošetřujeme využitím regulárních výrazů. Na výpisu zdrojového kódu 8 je vyobrazen regulární výraz, který nám zaručí, že budou vloženy maximálně 3 proměnné, které mohou být rozděleny čárkou s využitím znaků a-z.

```
<%String prohlizec=request.getHeader("User-Agent");%>
<form name="pocet_funkci" action=optimalizace2.jsp method="POST">
<h3>Zadej pocet funkci: <br><br>
<%if(prohlizec.matches(".*\\bSafari\\b.*")&&!prohlizec.matches(".*\\bChrome\\b
    .*"))
{%>
```

```
<input type="text" name="funkce" value="" size="20" style="width: 50px; padding
    : 2px" required maxlength="1" onkeypress='return event.charCode >= 49 &&
    event.charCode <= 57'/><br><br>
Zadej pocet promennych:</h3>
<input type="text" name="promenne" value="" size="20" style="width: 50px;
    padding: 2px" required maxlength="1" onkeypress='return event.charCode >=
    49 && event.charCode <= 57'/><br><br>
<%}
else
{%>
<input type="number" name="funkce" value="" min="1" step="1" max="9" size="20"
    style="width: 50px; padding: 2px" required maxlength="1" /><br><br>
Zadej pocet promennych:</h3>
<input type="number" name="promenne" value="" min="1" step="1" max="9" size="20
    " style="width: 50px; padding: 2px" required maxlength="1"/><br><br>
<%}%>
<input type="submit" value="Pokracovat" name="funkce_tl"/>
</form>
```

Výpis 7: Ukázka zdrojového kódu z optimalizace.jsp



Obrázek 14: JSP - Eliminace košíku pro optimalizace

```
<input type="text" name="funkce<%=i %>" value="" size="20" data-toogle="tooltip
    " title="Vloz maximalne 3 promenne a-z, ktere mohou byt rozdeleny carkou (
    napr.: a,b,c nebo abc)." style="width: 50px; padding: 2px" required
    maxlength="5" pattern="[a-z](([\0|\s,]*[a-z]){0,2})" />
```

Výpis 8: Ukázka využití regulárního výrazu

9.4 Eliminace košíku pro počítání počtu řešení a eliminace minikošíku - část SW na serveru

Stavba softwaru je stejná jako u eliminace košíku pro optimalizaci s úpravami pro určitý algoritmus. Eliminace minikošíku rozděluje jiným způsobem funkce do košíků, ale druhá fáze je stejná jako tomu bylo u eliminace košíku pro optimalizaci. Tento výpočet probíhá ve třídě `MinikosikyServlet.java` umístěné v java package „eu.aruo.program“.

Eliminace košíku pro počítání počtu řešení stejně rozděluje košíky, ale liší se výpočet funkcí z košíků a druhá fáze. Provádí se totiž jiné matematické operace a tento algoritmus má na vstupech pouze nuly a jedničky (tedy splnitelné, či nesplnitelné). Tento výpočet probíhá ve třídě `PocetReseniServlet.java` umístěné v java package „eu.aruo.program“.

9.5 Eliminace košíku pro počítání počtu řešení, eliminace minikošíku a pomocí pravděpodobnostních sítí - webová část

Webová část SW pro ostatní algoritmy je řešena opět obdobným způsobem (je zachována stejná struktura souborů). Ovšem u každého algoritmu jsou vstupy hlídány podle jejich potřeb. To se týká i zobrazování výsledků, kde je vždy stránka uzpůsobena typu řešeného algoritmu.

9.6 Eliminace košíku pomocí pravděpodobnostních sít - část SW na serveru

Stavba softwaru je opět podobná, ale zde nepočítáme s typem integer, nýbrž s typem double. Pravděpodobnostní sítě využívají tedy i své vlastní objekty umístěné v java package „eu.aruo.pravdepodobnost_data“. Zde jsou rozdíly oproti jiným algoritmům největší. Funkce do košíku se rozřazují jiným způsobem a také výpočet a druhá fáze probíhá jinak, i když podobně.

10 Generování PDF

K vytvoření PDF nám slouží třída `ExportServlet`, která se nachází v java package „eu.aruo.program“. Z webu se nám zde odesílá, jaký typ metody právě uživatel použil a aktuální čas. Aktuální čas je využit pro název souboru PDF a označení metody pro následné přesměrování zpět na webovou stránku.

K samotnému vytvoření PDF slouží metoda `vytvor_dokument`. Protože používáme sdílený server, musela jsem si nejprve zjistit od poskytovale absolutní cestu k mé složce na serveru. Po zjištění jsem již mohla vytvořit soubor ve své složce, které jsem ještě musela nastavit práva pro zápis. Ukázku tvorby souboru vidíme na výpisu zdrojového kódu 9.

```
String nazev = "/users/klienti/aruo.eu/aruo.eu/Soubory/"+cas+".pdf";
      PdfWriter.getInstance(document, new FileOutputStream(nazev));
        document.open();
```

Výpis 9: Ukázka kódu z metody „vytvor_dokument“

Po výpočtu metody jsme důležité údaje potřebné pro vygenerování PDF vložili do třídy `data.java` umístěné v package „eu.aruo.data“. Odsud čerpáme výsledky, které si sázíme procházením do PDF souboru. K samotnému vytvoření PDF se využívá externí knihovna s názvem itextpdf (bližší informace k této knihovně nalezneme na webu knihovny [9]).

Mapování servletů využívá více webových stránek právě tuto třídu. Ukázku mapování servletů vidíme na výpisu kódu 10. Toto mapování má název `web.xml`, funguje jako rozhraní mezi webovou částí a serverovou části aplikace a je uloženo ve složce WEB-INF u webové části softwaru.

```
<servlet>
    <description></description>
    <display-name>optimalizace7</display-name>
    <servlet-name>optimalizace7</servlet-name>
    <servlet-class>eu.aruo.program.ExportServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>optimalizace7</servlet-name>
    <url-pattern>/optimalizace7</url-pattern>
 </servlet-mapping>
 <servlet>
    <description></description>
    <display-name>eliminace_mini7</display-name>
    <servlet-name>eliminace_mini7</servlet-name>
    <servlet-class>eu.aruo.program.ExportServlet</servlet-class>
 </servlet>
```

```
<servlet-mapping>
   <servlet-name>eliminace_mini7</servlet-name>
   <url-pattern>/eliminace_mini7</url-pattern>
</servlet-mapping>
```

Výpis 10: Ukázka mapování servletu

11 Uživatelské rozhraní

Po načtení stránky `www.aruo.eu` se jako první zobrazí stránka s označením `index.jsp`. Nejdůležitějším prvkem pro orientaci na webové stránce je navigační menu lišta. Úvodní stranu spolu s horní navigační lištou vidíme na obrázku 15. Pomocí horní navigace lze vybrat, kterou metodu eliminace košíku chceme počítat.

Po vybrání metody, s kterou chceme pracovat, uživateli opět vzniká více možností. Může se rozhodnout, zda si zadá vstupy své, nebo se podívá na již zadané řešení. K tomuto úkonu slouží tlačítka umístěná na stránce s popisem „zadané řešení“ (ukázku vzhledu webové stránky vidíme na obrázku 16). Na této stránce uživatel zvolí počet funkcí a počet proměnných. Aby uživatel věděl, jaké hodnoty může vkládat, je na políčku umístěn tooltip s popisem.

Podle zadaných hodnot se na další straně vygenerují políčka pro zadávání, opět všechny políčka obsahují tooltip. Všechny vkládané hodnoty jsou kontrolovány pomocí javasciptu. Protože vyplňování hodnot všech funkcí bývá někdy pracné, a nejde nám o konkrétní výpočet, můžeme si v dolní části vybrat, zda chceme vygenerovat hodnoty do pole, nebo si hodnoty zadáme sami. Vzhled tohoto okna vidíme na obrázku 17. Po stisknutí tlačítka „Pokračovat“ se stránka přesměruje dále a okno se vysází podle zadaných rozsahů proměnných.

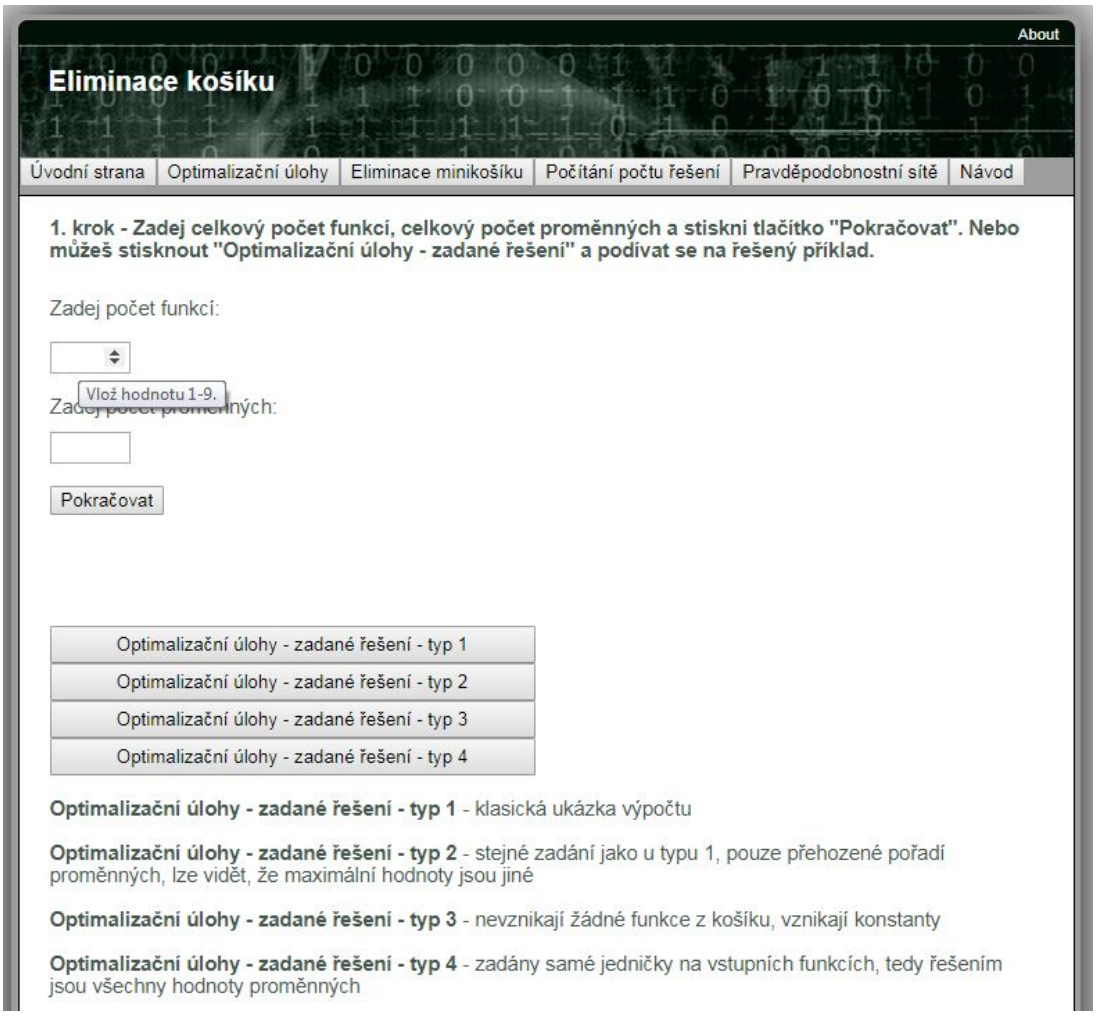
Než se další webová stránka vysází, probíhá kontrola správnosti zadání - například, zda jsou využity všechny proměnné. Při špatném zadání se zobrazí varovné pole, které vidíme na obrázku 18, a přesměruje nás zpět na stránku, kde bylo špatné zadání. Na stránce `optimalizace3.jsp` uživatel vyplní (nebo si nechá vygenerovat) hodnoty proměnných a stiskne tlačítko „Pokračovat“. Po stisku tlačítka se všechny zadané data odesílají na server ke zpracování (probíhá výpočet všech hodnot). Po úspěšném výpočtu se uživateli otevře stránka `optimalizace4.jsp`, kde vidí popis s rozdělením košíků. Opět pokračuje stiskem tlačítka „Pokračovat“.

Další strana s označením `optimalizace5.jsp` se otevře s výpočty pro první fázi. Pro bližší informace může uživatel kliknout na požadovaný řádek, nebo stiknout tlačítko s popisem „Podrobnosti“. Vyskočí malé vyskakovací okno s výpočtem (vzor tohoto okna vidíme na obrázku 19). Dále uživatel pokračuje stisknutím tlačítka „Pokračovat“.

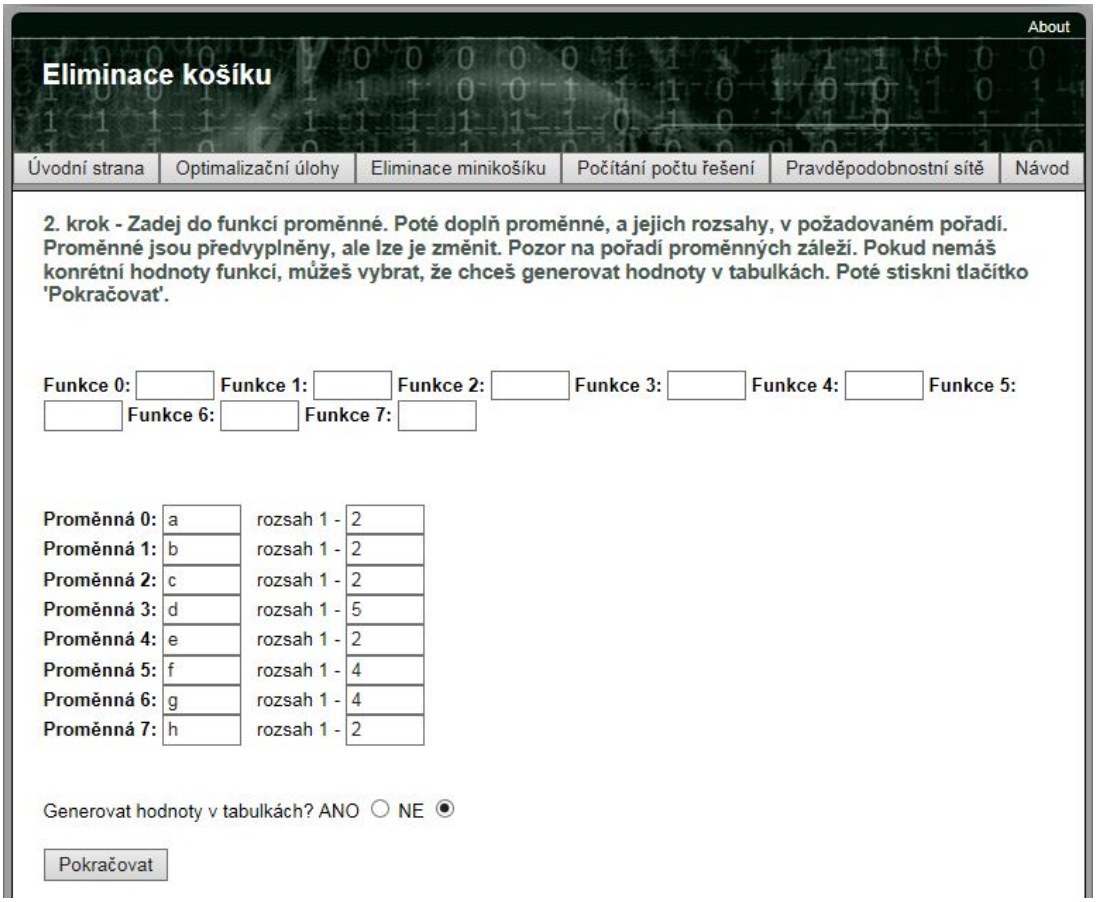
Poslední strana, která se týká výpočtu má označení `optimalizace6.jsp`. Vysází si tabulky s výpočty a výslednými hodnotami, které jsou šedě podbarveny. Pokud má uživatel zájem a chce se k výsledkům příkladu později vrátit, může si vygenerovat PDF s hodnotami výpočtu (tato možnost je u všech metod kromě pravděpodobnostních sítí). Pro tento úkon slouží tlačítko „Vygeneruj PDF“. Po jeho stisknutí se opět pošlou údaje na server, kde dojde ke zpracování a vytvoření PDF souboru, poté se soubor nabídne uživateli ke stažení (samotné vygenerování PDF bude v této práci popsáno později).

Tato struktura stránek je využívána u všech metod. Stránky u jiných metod jsou modifikovány podle potřeb určité metody. Například u pravděpodobnostních sítí se nezadávají sítě omezení, ale pravděpodobnostní sítě. Také zadaní hodnot není v integerech, ale v datovém typu double a takto bychom mohli jmenovat dále.

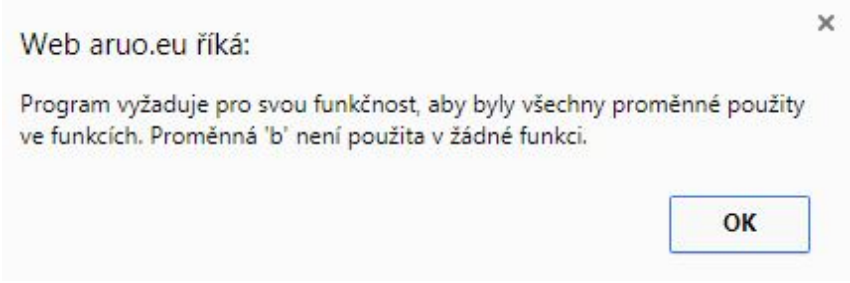
Obrázek 15: Ukázka vzhledu webové stránky



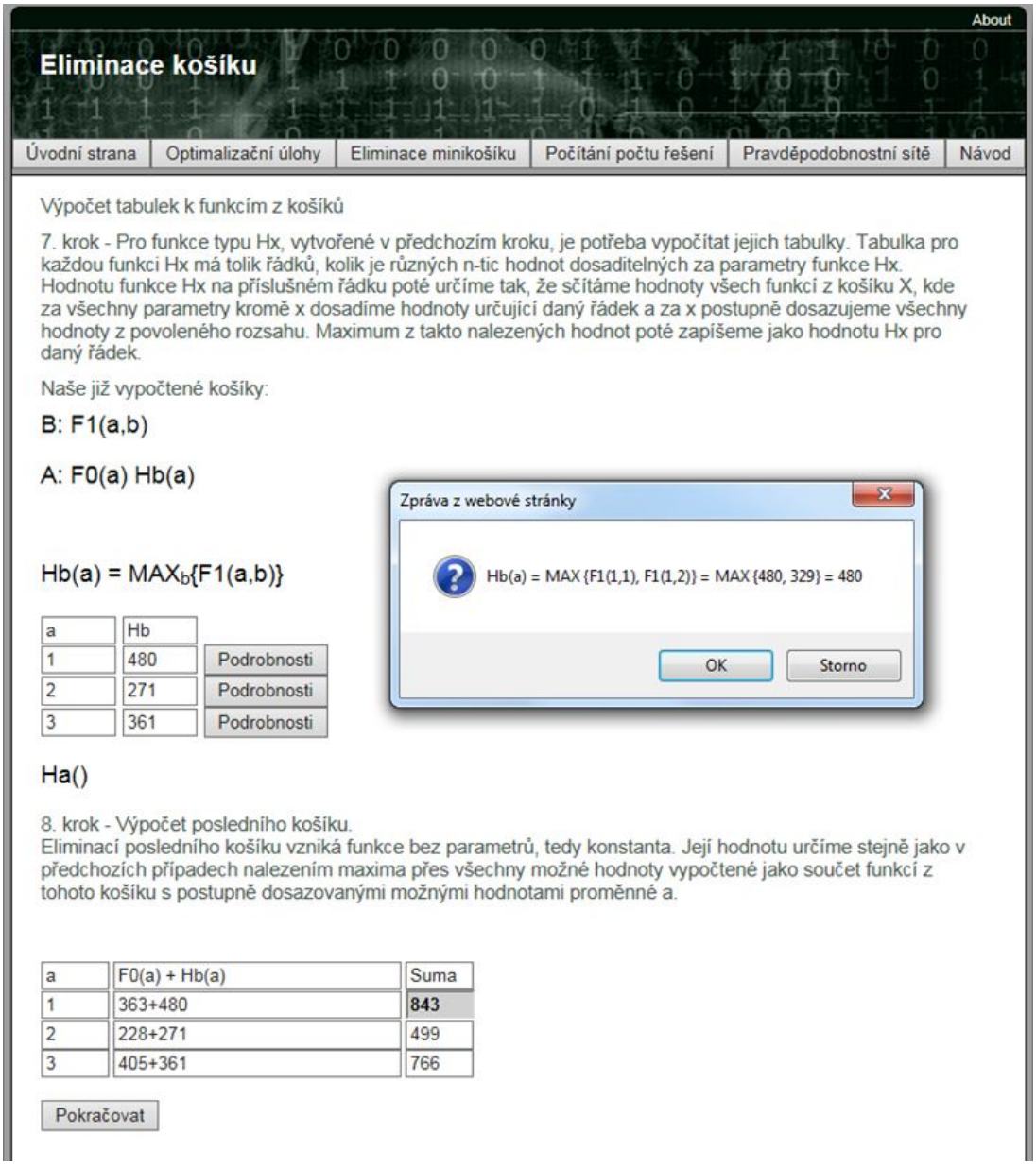
Obrázek 16: Ukázka vzhledu webové stránky - optimalizační úlohy



Obrázek 17: Ukázka vzhledu webové stránky - optimalizační úlohy - 2. okno



Obrázek 18: Ukázka vzhledu webové stránky - chybová hláška



Obrázek 19: Ukázka vzhledu webové stránky - vyskakovací okno s podrobnostmi výpočtu

12 Nasazení aplikace

Pro nasazení této aplikace je zapotřebí server Tomcat v7.0 (je možné, že aplikace poběží i na jiné verzi Tomcatu, ovšem není to nevyzkoušeno). Existují dva způsoby jak si aplikaci spustit.

Prvním způsobem je využít webovou stránku. Pro testování byla aplikace nasazena na webovou stránku `www.aruo.eu`. Není mnoho českých poskytovatelů, kteří nabízejí JSP webhosting s nějakým serverem Tomcat. Vybrala jsem si tedy poskytovatele G-HOSTING, který nabízí za celkem přijatelnou cenu sdílený Tomcat server. Pro připojení ke své složce na serveru je využívána aplikace `WinSCP`. Po přihlášení můžeme upravovat soubory a nahrávat nové verze aplikace.

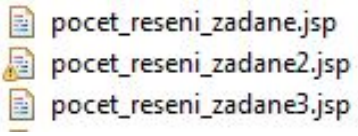
Aplikace pro nasazení na web musí mít název `ROOT.war`, aby se hned po načtení stránky aplikace spustila. Pro vygenerování *.war souboru slouží ve vývojovém prostředí Eclipse funkce. Stačí v horním menu Eclipse zvolit File\Export..\ a zde vygenerovat *.war soubor s názvem ROOT. Poté tento soubor pomocí WinSCP nahrajeme do kořenové složky. Nyní můžeme spustit aplikaci přes webový prohlížeč zadáním adresy `www.aruo.eu`.

Pro zdatnější uživatele, kteří mají nainstalováno vývojové prostředí Eclipse a server Tomcat, si mohou aplikaci spustit u sebe v počítači pomocí localhostu.

13 Vytvoření nového vzorového příkladu

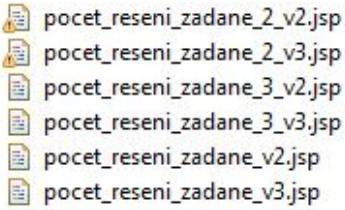
Vytvoření nového vzorového příkladu je velmi jednoduché, protože webové stránky pro zadání a výpočet jsou rozděleny. Tedy stačí upravit první 3 kroky zadání. Popíšeme si přidání zadání pro metodu počítání počtu řešení. U ostatních metod je vytváření stejné, jen u pravděpodobnostních sítí je navíc přidáno pole s názvy pravděpodobnostních sítí.

Doporučuji použít jako základ vždy JSP soubory, které vidíme na obrázku 20. U ostatních algoritmů to bude totožné, vždy používejte jako vzor JSP soubor, který nemá v názvu „v2“, či nějakou jinou verzi.



Obrázek 20: Vzor pro tvorbu zadaných řešení

Tyto soubory si zkopírujeme a vložíme do stejné složky, ve které se nacházíme, ovšem je důležité soubory přejmenovat, například tak, že v každém názvu upravíme verzi. Strukturu souborů dalších dvou zadaných příkladů vidíme na obrázku 21.



Obrázek 21: Vzor pro tvorbu zadaných řešení - nová verze

Po vytvoření souborů měníme obsah v nově vytvořených souborech. U původního souboru `pocet_reseni.jsp` změníme, kde se má po stisknu tlačítka stránka přesměrovat. Podle vzorového kódu 11 tedy změníme první řádek `action=pocet_reseni_zadane2.jsp` na `action=pocet_reseni_zadane2_v2` (nebo název, kterým jste si pojmenovali další potřebné okno). V tomto okně už jen změníme u zadávacích polí jejich pevnou hodnotu, tedy změníme parametr value. V našem vzorovém příkladě máme value=6, můžeme ho tedy libovolně měnit v povoleném rozsahu. Je zapotřebí si hlídat povolené hodnoty, protože zde hodnoty nezadáváme z klávesnice.

```
<form name="pocet_funkci" action=pocet_reseni_zadane2.jsp method="POST">
   <h3><b> 1. krok - Zadej celkovy pocet funkci, celkovy pocet promennych a
      stiskni tlacitko "Pokracovat".</b></h3><br>
   <h3>Zadej pocet funkci: <br><br>
```

```
    <input type="number" name="funkce" value="6" size="20" style="width: 25px;
        padding: 2px" readonly/><br><br>
    Zadej pocet promennych:</h3>
    <input type="number" name="promenne" value="6" size="20" style="width: 25px;
         padding: 2px" readonly/><br><br>
    <input type="submit" value="Pokracovat" name="funkce_tl"/>
</form>
```

Výpis 11: Ukázka zdrojového kódu z pocet_reseni_zadane.jsp

Po uložení následuje úprava dalšího námi zkopírovaného a přejmenovaného souboru. Opět změníme řádek s definicí, kam se má stránka přesměrovat po stisknutí tlačítka, tedy řádek začínající `<form name=...` . Dále je zapotřebí nadefinovat, jak budou vypadat vstupní funkce, pořadí proměnných a jejich rozsahy. Které řádky zde měníme, vidíme na ukázce kódu 12. Můžeme libovolně měnit délku pole, ovšem musí se shodovat s nastavením předchozí stránky. Pokud jste zadali počet funkcí 6, musí mít pole délku 6. Po změně těchto 3 polí již soubor uložíme a pokračujeme na poslední námi vytvořený soubor.

```
String []pole_funkce = new String[6];
pole_funkce[0]="a";
pole_funkce[1]="ab";
pole_funkce[2]="ac";
pole_funkce[3]="bcf";
pole_funkce[4]="abd";
pole_funkce[5]="fg";

String []pole_prome = new String[6];
pole_prome[0]="a";
pole_prome[1]="c";
pole_prome[2]="b";
pole_prome[3]="f";
pole_prome[4]="d";
pole_prome[5]="g";

int []pole_konce = new int[6];
pole_konce[0]=4;
pole_konce[1]=3;
pole_konce[2]=5;
pole_konce[3]=6;
pole_konce[4]=2;
pole_konce[5]=3;
```

Výpis 12: Ukázka zdrojového kódu z pocet_reseni_zadane2.jsp

Zbývá poslední námi vytvořený soubor, který byl vytvořen zkopírováním souboru `pocet_reseni_zadane3.jsp` . Tentokrát neměníme, kde se má stránka přesměrovat po stisku tlačítka, tedy řádek začínající `<form name=` **necháváme beze změny**. Nyní nám již zbývá zadat hodnoty funkcí. Využívají se pole s označením „pole_f0, pole_f1,..“, u kterých stačí změnit délku pole (podle toho, jaké jsme zadali rozsahy proměnných a kolik proměnných je ve funkci) a jejich konkrétní hodnotu. Využívá se zde klasická java, takže se nebojte si práci usnadnit, a naplnit pole třeba cyklem for. Jak vypadá pole, které máme měnit, vidíme na ukázce kódu 13. Pokud chcete zadávat hodnoty ručně, tak je to trochu pracnější. Zdrojový soubor, ze kterého jsme tento soubor vytvořili, je připraven na maximálně 6 polí (tedy 6 funkcí), jednoduše můžeme přidat další, až do počtu 9. Stačí přidat další vkladání polí, jak vidíme na ukázce kódu 14. Tedy pro 7. pole by se řádek vložil za úkazku s podmínou `if(i==6)`, pouze se zkopíruje a upraví zadávací políčko na proměnnou a změnil by se atribut value na proměnnou, kde máme vytvořeno další pole. Pokud bychom chtěli zachovat stejný styl, tak by se pole jmenovalo `pole_f6`.

```
int []pole_f0 = new int[4];
pole_f0[0]=0;
pole_f0[1]=0;
pole_f0[2]=1;
pole_f0[3]=1;
```

Výpis 13: Ukázka zdrojového kódu z pocet_reseni_zadane3.jsp

```
if(i==4){%>
<input type="number" name="<%="funkce"+i+"radek"+j+"sloupec"+k%>" value="<%=
    pole_f4[j-1] %>" size="20" style="width: 30px; padding: 2px" required
    readonly/>
<%}
if(i==5){%>
<input type="number" name="<%="funkce"+i+"radek"+j+"sloupec"+k%>" value="<%=
    pole_f5[j-1] %>" size="20" style="width: 30px; padding: 2px" required
    readonly/>
<%}
```

Výpis 14: Ukázka zdrojového kódu z pocet_reseni_zadane3.jsp

Když máme vytvořeny a upraveny všechny 3 soubory, zbývá nám upravit úvodní soubor, v našem případě `pocet_reseni.jsp`. Je zapotřebí přidat tlačítko pro „spuštění“ našeho nového zadaného příkladu. Stačí zkopírovat jeden z řádků pro tlačítko (viz ukázka zdrojového kódu 15) a změnit mu parametry „id“ a „value“ s názvem našeho nového prvního vytvořeného souboru,

v našem případě `pocet_reseni_zadane.jsp`. Spusťte si program na localhostu a zkontrolujte, zda se vše zobrazuje, jak má. Po odzkoušení již nic nebrání vygenerovat soubor `ROOT.war` (jak vygenerovat tento soubor popisuji v kapitole 12) a nahrát na server.

```
<input id="inp6" type="button" value="Pocitani poctu reseni - zadane reseni -
    typ 1" style="width: 350px; padding: 4px" onclick="location.href='
    pocet_reseni_zadane.jsp';" />
<input id="inp7" type="button" value="Pocitani poctu reseni - zadane reseni -
    typ 1" style="width: 350px; padding: 4px" onclick="location.href='
    pocet_reseni_zadane_v3.jsp';" />
```

Výpis 15: Ukázka zdrojového kódu z pocet_reseni.jsp

14 Závěr

Cílem této práce bylo vytvoření serveru pro podporu výuky předmětu ARUO, na kterém si studenti budou moci spustit výpočet vybraných algoritmů z oblasti constraint processing a jejich metody eliminace košíků. Nebylo úkolem naprogramovat algoritmus efektivně, ale tak, aby uživatel byl schopen si zobrazit postupné kroky výpočtu a pochopil fungování algoritmu.

Úkolem bylo vytvoření takové aplikace, kde si uživatel může zvolit své vstupy, a na těch mu bude algoritmus vysvětlen, případně použít již vytvořené vstupy pro prohlédnutí funkčnosti bez zadání vstupu uživatelem.

V první části jsme definovali co to vůbec Constraint processing je, blíže si vysvětlili, co jsou optimalizační problémy, a přiblížili metodu eliminace košíku. Eliminace košíku není vysvětlena pouze pro optimalizaci, ale také pro použití počítání počtu řešení, pro eliminace minikošíku, či pro použití metody pravděpodobnostních sítí.

Druhá část je zaměřena na samotnou implementaci algoritmů pro eliminaci košíků. Prošli jsme si jak funkční a nefunkční požadavky, tak využití knihovny itexpdf pro generování PDF. Ukázali jsme si, jak aplikace funguje z pohledu uživatele. Na závěr této části jsme popsali, jakým způsobem lze aplikaci nasadit a jak přidat předem zadaný příklad.

Při původní implementaci programu bylo využito java apletu, ovšem po konzultaci a uvědomění si, že java aplet nemusí fungovat na přenosných zařízení jako jsou mobilní telefony, či tablety se implementace změnila na webovou aplikaci klient-server. Tento typ aplikace již funguje na přenosných zařízení, které se v dnešní době čím dál více používají, a uživatele mají přístup k webu odkudkoli.

Během implementace webové části aplikace nastal problém s prohlížeči. V prohlížeci Firefox nefungovalo vše tak, jak v prohlížeči Safari, a naopak. Proto se přistoupilo k patřičným úpravám, které jsou popsány v kapitole 9.3.

Celá aplikace je řešena tak, aby bylo jednoduché vložit pevná vzorová zadání pro různé algoritmy, ale také pro možnost rozšíření o další metody eliminace košíku, například pro metodu ELIM-BEL z oblasti pravděpodobnostních sítí. Také by se mohly rozšířit zadávané vstupy.

Struktura souboru je u všech algoritmů podobná, aby bylo jednoduché se rychle orientovat či upravovat software. Popisy na webových stránkách jsou řešeny přes JSP soubory a HTML, takže je možnost jednoduše je upravit podle potřeby.

Aplikace prozatimně běží na webové stránce „www.aruo.eu“, kde si aplikaci můžete plně vyzkoušet či ji využívat. Doufám, že tato aplikace bude přínosem a pomůže studentům ve studiu.

Literatura

[1] DECHTER, Rina. Constraint processing. San Francisco: Morgan Kaufmann Publishers, c2003. ISBN 1-55860-890-7.

[2] BARTÁK, Roman. Programování s omezujícími podmínkami [online]. , 7 [cit. 2016-06-01]. Dostupné z: http://kti.mff.cuni.cz/ bartak/podminky/lectures/lecture01.pdf

[3] BARTÁK, Roman. Constraint programming [online]. [cit. 2016-06-20]. Dostupné z: http://kti.mff.cuni.cz/ bartak/podminky/lectures/CSP-lecture09eng.pdf

[4] DECHTER, Rina. Constraint networks [online]. University of California [cit. 2016-06-01]. Dostupné z: http://www.ics.uci.edu/ dechter/publications/r17-survey.pdf

[5] POOLE, David L. a Alan K. MACKWORTH. Artificial intelligence: foundations of computational agents. Cambridge: Cambridge University Press, 2010. ISBN 978-0-521-51900-7.

[6] PEARL, Judea a Stuart RUSSELL. Bayesian networks [online]. , 13 [cit. 2016-06-22]. Dostupné z: https://people.eecs.berkeley.edu/ russell/papers/hbtnn-bn.pdf

[7] PEARL, Judea. Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, 2014.

[8] FRIEDMAN, Nir; KOLLER, Daphne. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. Machine learning, 2003, 50.1-2: 95-125.

[9] IText [online]. [cit. 2016-06-27]. Dostupné z: http://itextpdf.com/

A Obsah CD

Následující tabulka popisuje soubory a složky na CD a jejich popis.

Soubor nebo složka	**Popis**
Aplikace	Aplikace s dokumentací vygenerovanou pomocí JavaDocu (Dokumentace je umístěna ve složce \ROOT\doc)
LEG0012.pdf	Text diplomové práce
TeX	Zdrojové soubory pro LaTeX, včetně použitých obrázku a úryvků zdrojových kódů
ROOT	Potřebné soubory a složky pro nasazení na server
navod.pdf	Návod pro uživatele

Tabulka 13: Obsah CD

B Uživatelská příručka - Návod

B.1 Úvodní strana

Po načtení webové stránky www.aruo.eu se nám zobrází stránka zobrazena na obrázku 22.

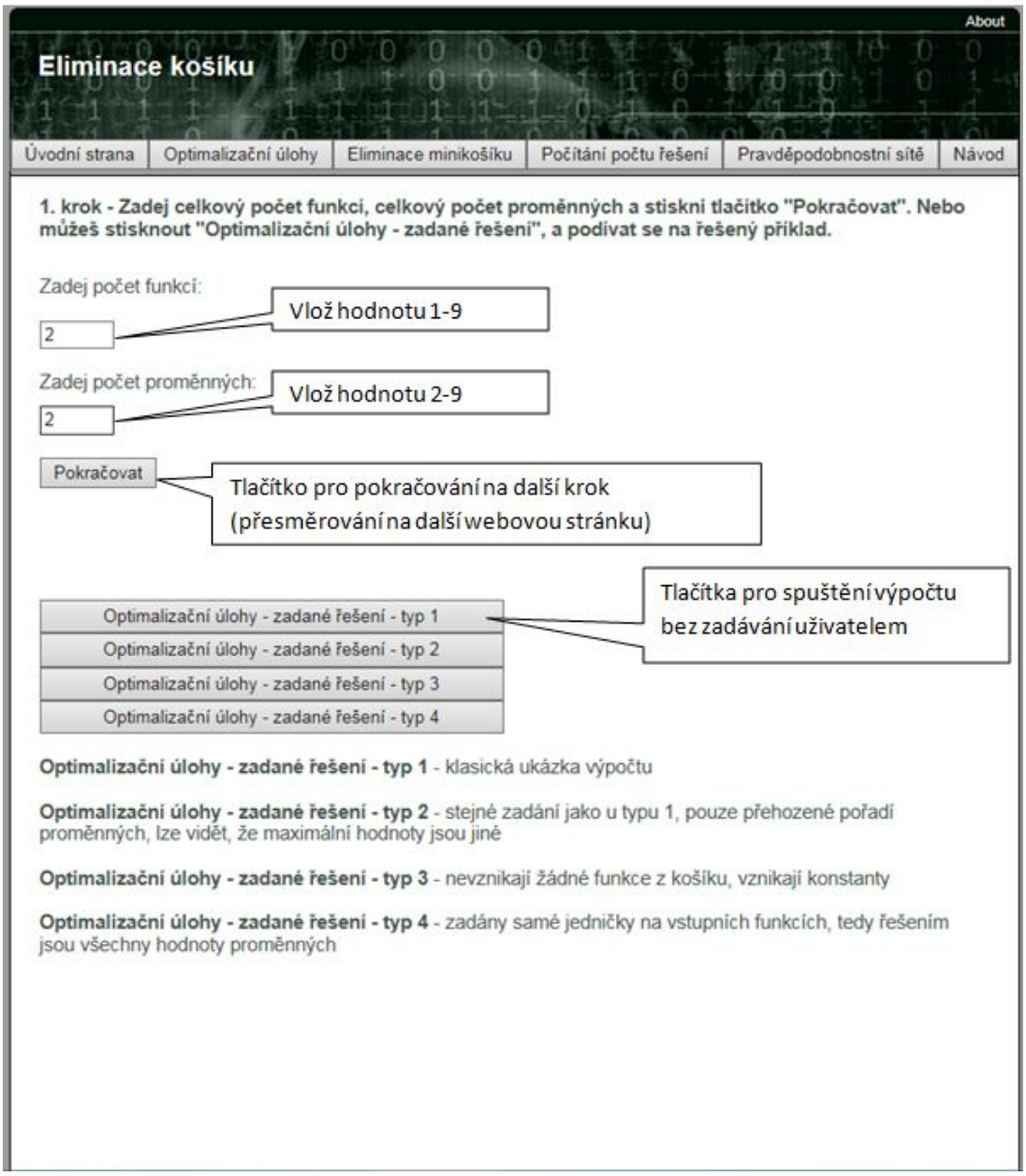
Obrázek 22: Uvodní strana

Vysvětlení základních části webové stránky:

1. Tlačítko pro spuštění požadované metody Eliminace košíku
2. Zobrazení návodu
3. Navigační panel s tlačítky
4. Obsah webové stránky, zde se budou zobrazovat veškeré výpočty

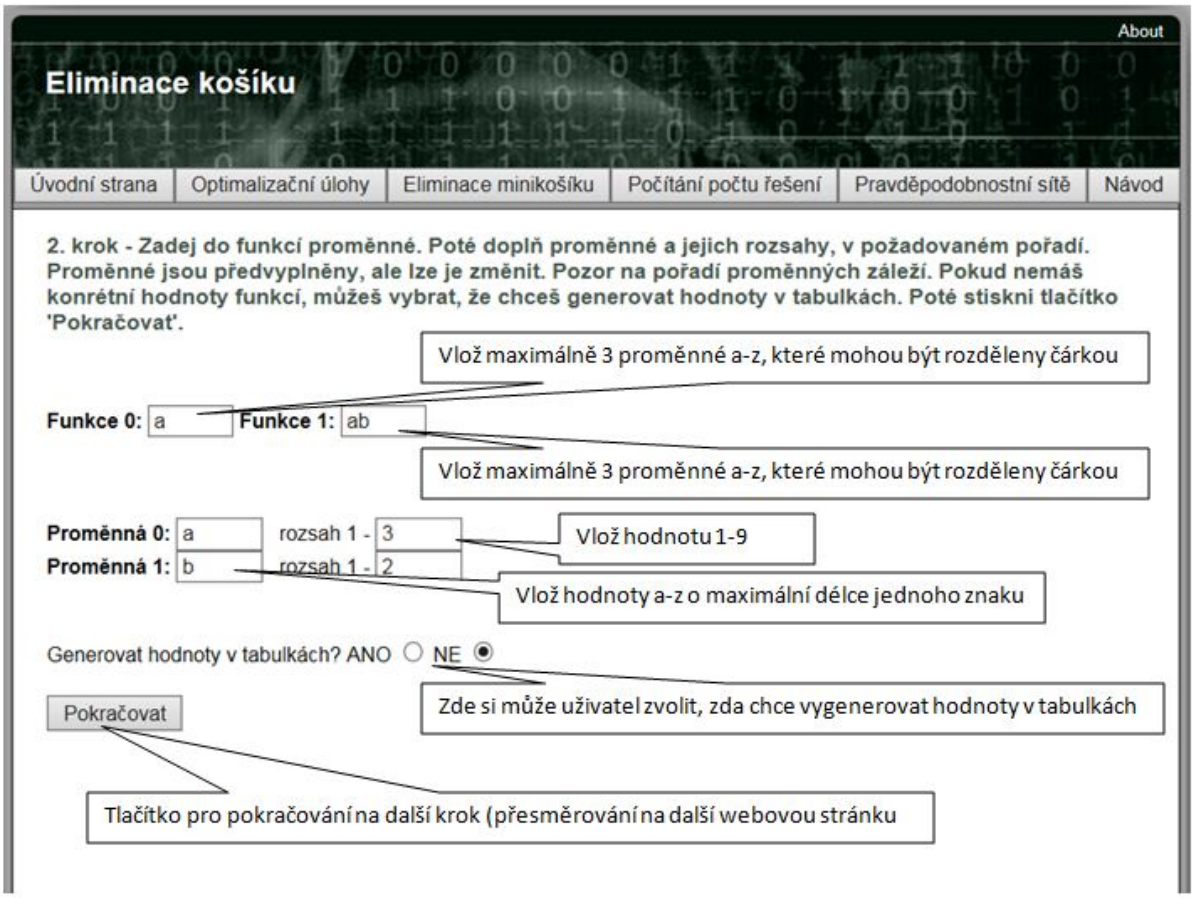
B.2 Optimalizační úlohy, Eliminace minikošíku, Počítání počtu řešení

Po zvolení jedné z metod eliminace košíku (mimo metodu pravděpodobnostních sítí) pomocí tlačitka v navigačním panelu se zobrazí stránka jako na obrázku 23. Funkčnost tlačítek a povolené hodnoty k zadání jsou zobrazeny na obrázku pomocí bublin. Zadávání je u všech třech metod obdobné, takže můžeme použít tento návod na metody optimalizačních úloh, eliminace minikošíku a počítání počtu řešení. Metodu pravděpodobnostních sítí najdeme dále v kapitole B.4.



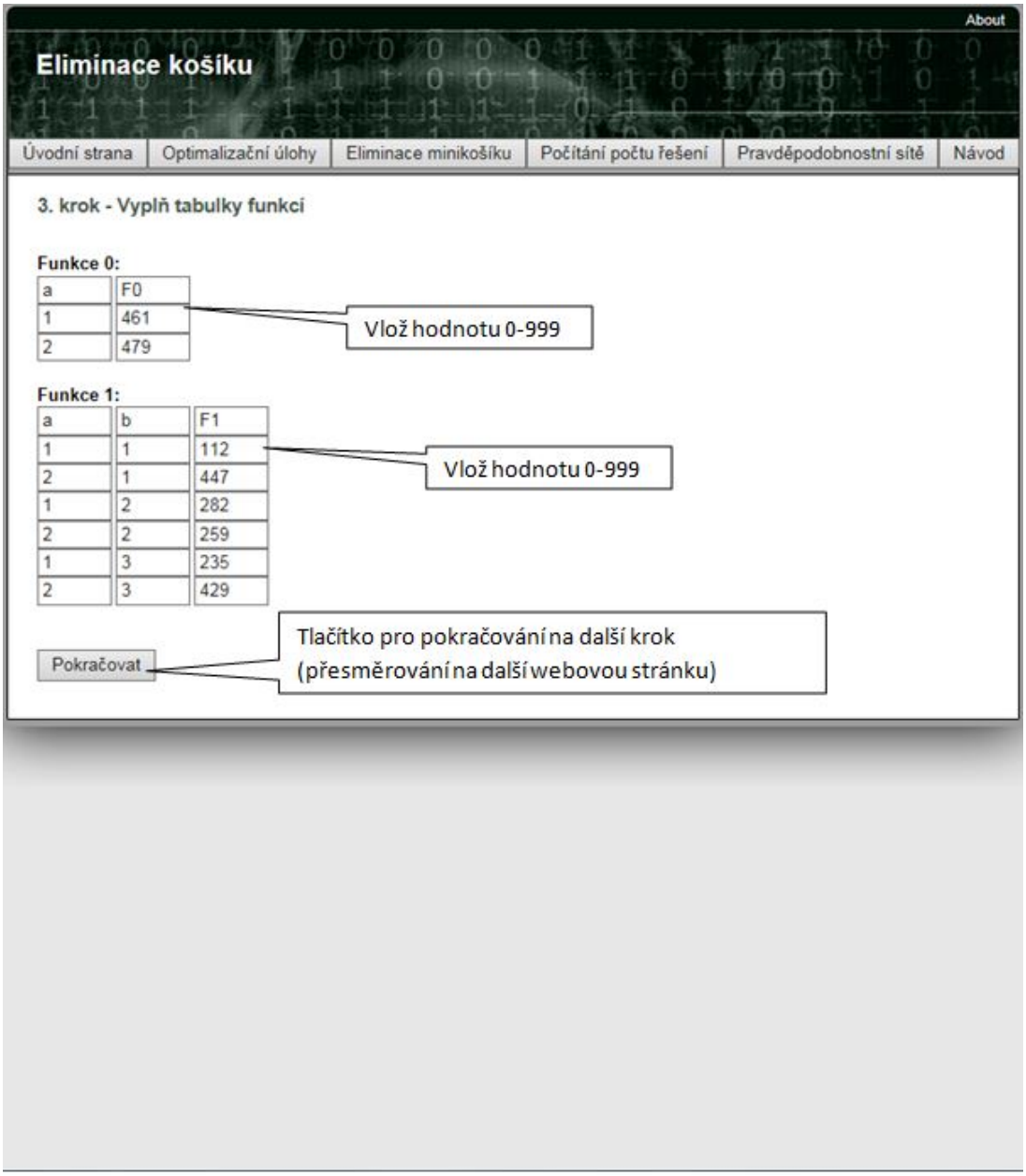
Obrázek 23: Strana 1 - zadávání počtu funkcí a proměnných

Po stisku tlačítka „Pokračovat“ se zobrazí stránka jako na obrázku 24. V bublinách v obrázku lze vidět popis, jaké hodnoty lze vkládat, či funkčnost dalších prvků.



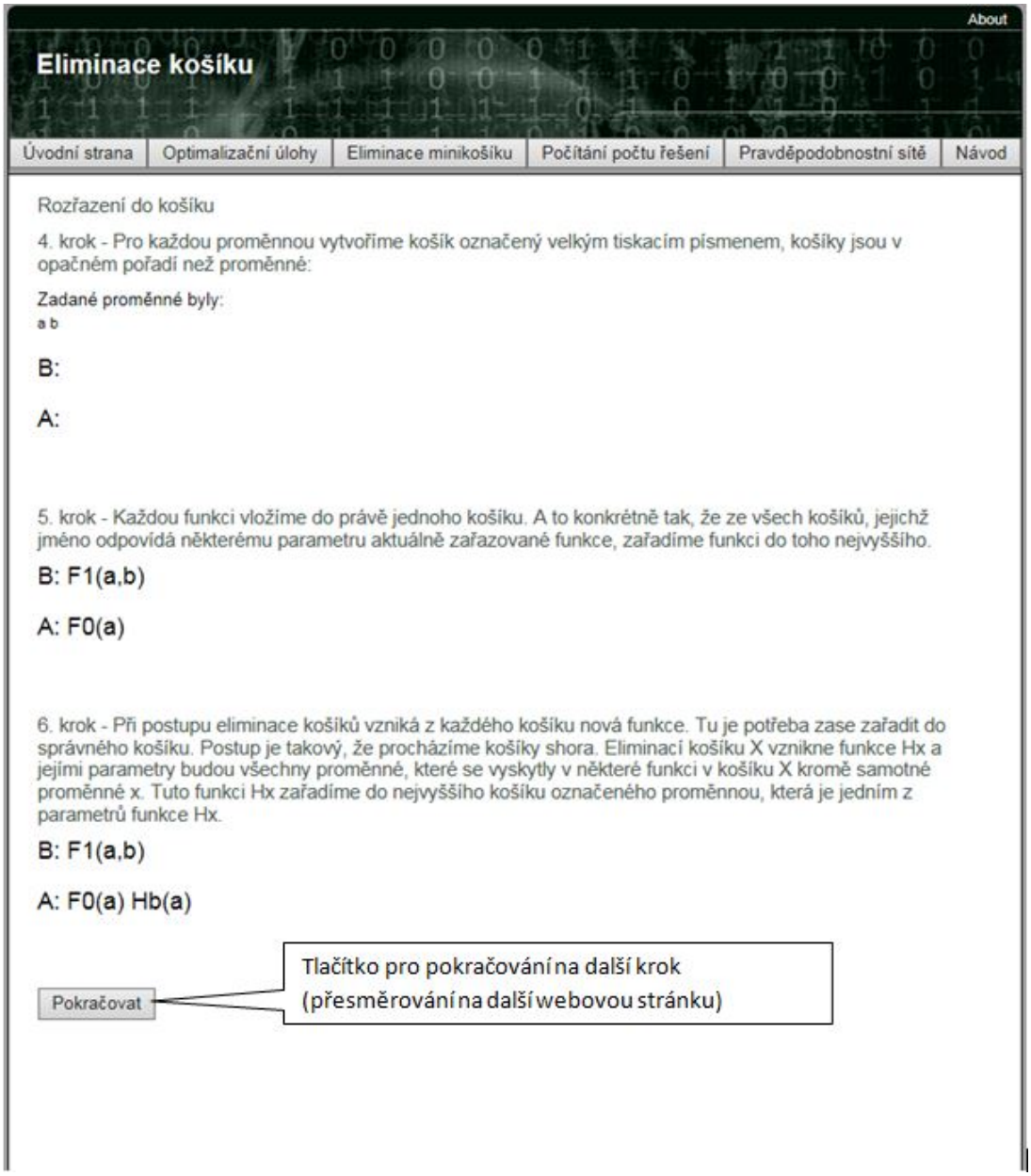
Obrázek 24: Strana 2 - zadávání funkcí, pořadí proměnných a jejich rozsahů

Po stisku tlačítka pokračování se zobrazí strana jako na obrázku 25. V bublinách v obrázku lze vidět, jaké hodnoty může uživatel vkládat. U Počítání počtu řešení je zde rozdíl, lze zadávat pouze hodnoty 0 nebo 1.



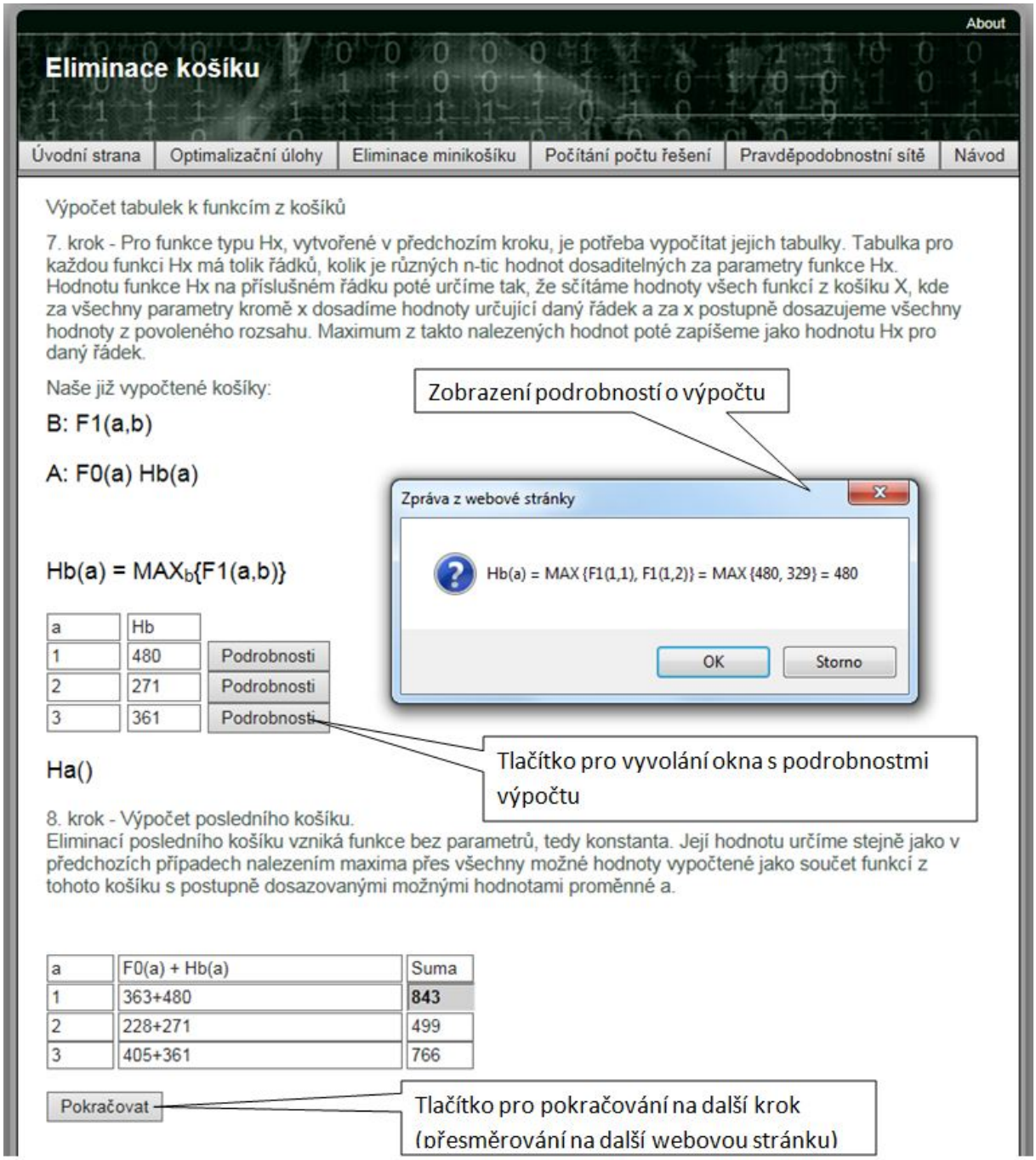
Obrázek 25: Strana 3 - zadávání hodnot funkcí

Po zadání všech hodnot a stisku tlačítka „Pokračovat“ se zobrazí stránka jako na obrázku 26. Od této strany se již nic uživatelem nezadává a zobrazuje se výpočet.



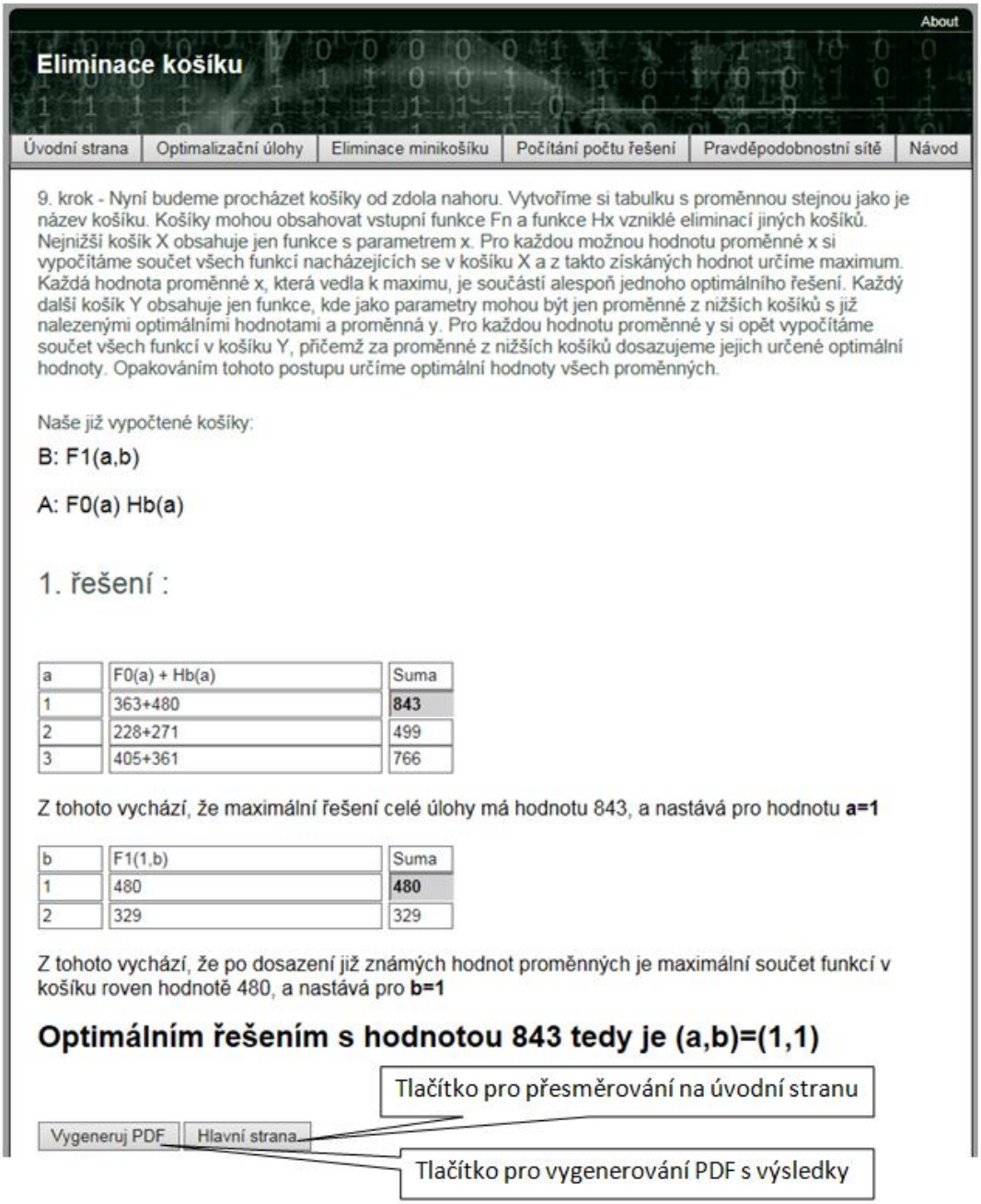
Obrázek 26: Strana 4 - první fáze výpočtu

Po prostudování stránky a stisku tlačítka „Pokračovat“ budeme přesmerování na stranu, která vypadá jako na obrázku 27. Bublinami v textu máme popsánu funkčnost tlačítek. Kliknutím na tlačítko podrobnosti nebo na řádek s výpočtem se zobrazí vyskakovací okno s podrobnostmi výpočtu.



Obrázek 27: Strana 5 - první fáze výpočtu

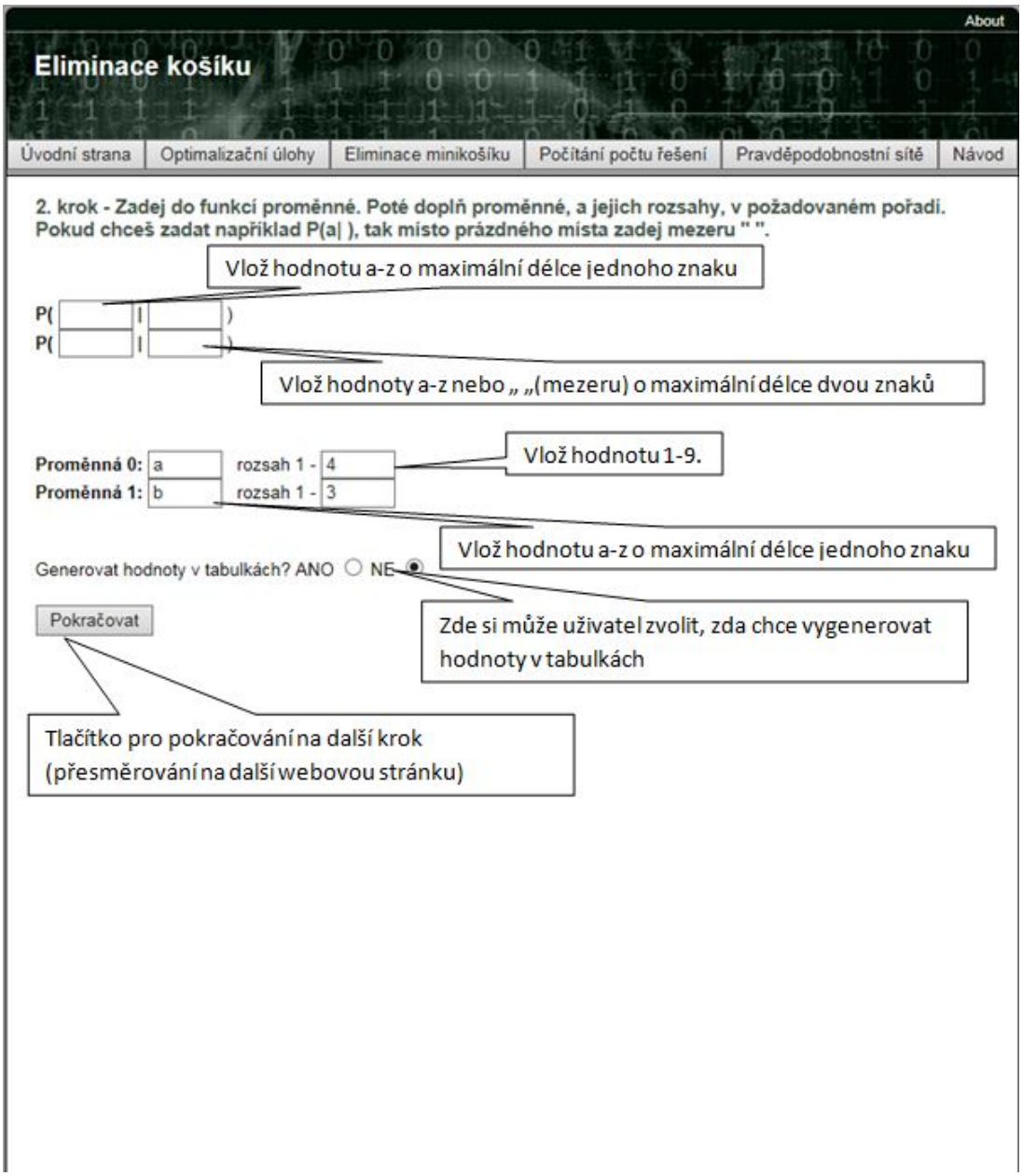
Po prostudání výpočtů na stránce a stisku tlačítka se dostaneme na stranu jako na obrázku 28. Zde se uživatel může přesměrovat na hlavní stranu, nebo si nechat vygenerovat PDF s výsledky pomocí tlačítka vygeneruj PDF. Po stisku tlačítka vygeneruj PDF se vytvoří PDF s výsledky, které si uživatel může stáhnout.



Obrázek 28: Strana 6 - druhá fáze výpočtu

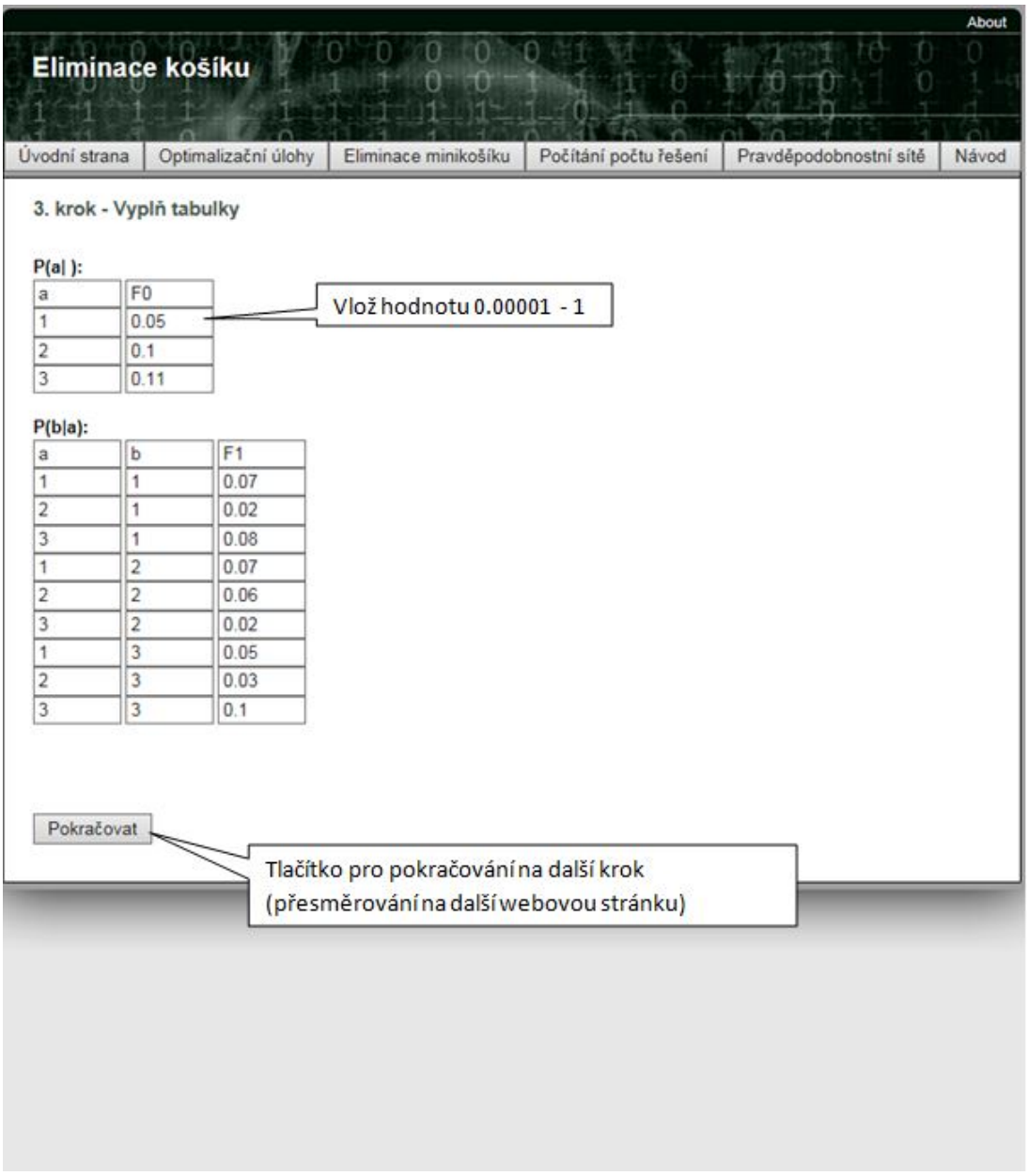
B.3 Pravděpodobnostní sítě

Po stisku tlačítka v horní navigaci s popisem pravděpodobnostních síti se zobrazí obdobné okno se stejnými podmínkami jako na obrázku 24. Po stisku tlačítka „Pokračovat“ se zobrazí okno již s jinými podmínkami než u ostatních metod, toto okno je vyobrazeno na obrázku 29.



Obrázek 29: Strana 2 - pravděpodobnostní sítě - zadání funkcí, pořadí proměnných a jejich rozsahů

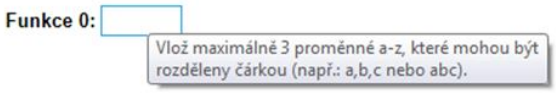
Po stisku tlačítka „Pokračovat“ se zobrazí stránka jako na obrázku 30. Po zadání všech hodnot, a stisku tlačítka pokračovat se stránky chovají stejně, jako v kapitole B.2 pouze s jinými popisy, či jiným zobrazením podrobností výpočtu.



Obrázek 30: Strana 2 - pravděpodobnostní sítě - zadání hodnot do tabulek

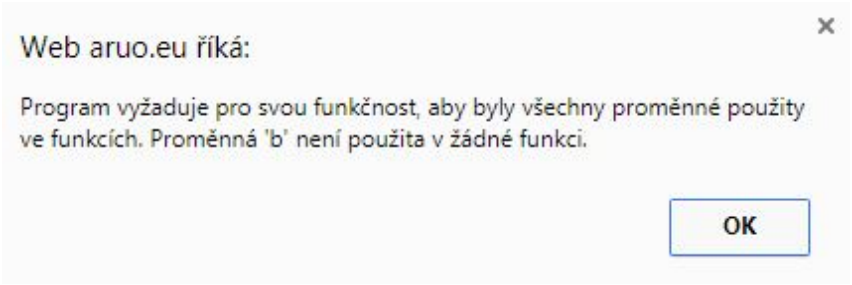
B.4 Chybové hlášky a vylepšení

Uživatel je schopen zjistit, jaké hodnoty lze zadat i bez tohoto návodu. Nad každým políčkem, kde je zapotřebí zadat hodnotu uživatelem, se nachází tzv. tooltip. Po najetí myši na vyplňovací políčko se zobrazí text s popiskem, co může uživatel zadat. Příklad tooltipu je zobrazen na obrázku 31.

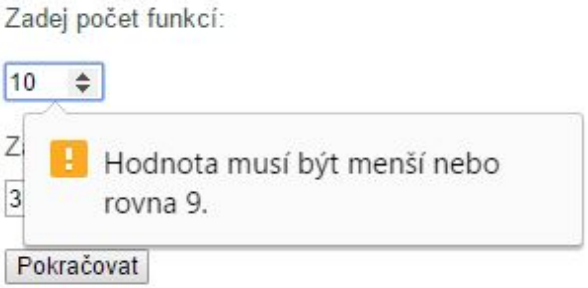


Obrázek 31: Tooltip

Při zadávání hodnot na webové stránce se provádí kontroly, aby uživatel nezadal špatnou hodnotu. Příklady zobrazení těchto ošetření je zobrazeno na obrázcích 32 a 33.



Obrázek 32: Chybová hláška



Obrázek 33: Upozornění přímo na webové stránce u políčka